1999

Algebraic completions of SBIBD

Julilati Utami
University of Wollongong

Algebraic Completions of SBIBD

A thesis submitted in fulfillment of the
requirements for the award of the degree

MSc (Honours)

from

UNIVERSITY OF WOLLONGONG

by

Juliati Utami

School of Information Technology and Computer Science
December 1999

Dedicated to
my parents Arijadi and S.Sendari Nitimidjoyo

Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

Juliati Utami
November 30, 1999

Abstract

We study the minimal set of vectors required to make the $(0,1)$ incidence matrix of an $SBIBD(v,k,\lambda)$. We devise algorithms to generate all possible vectors which could complete $SBIBD$ and to find minimal sets of such vectors. The 2-(7,3,1) and 2-(16,6,2) designs were studied to test our algorithms on the work of Greenhill and Street. We show the minimal defining set of the $SBIBD(31,15,7)$ constructed using the Paley difference set has 13 blocks. We give a minimal defining set with 19 blocks for the $SBIBD(31,15,7)$ constructed using the Hall difference set. We show the smallest minimal defining set of teh Hall SBIBD(31,15,7) has between 16 and 20 elements. We investigate the influence of the removal of vectors from minimal sets and discuss the relationship with secret sharing schemes.

Acknowledgements

I wish to express my gratitude to my supervisor, Professor Jennifer Seberry for her constant support and valued guidance. I must warmly thank to Professor Joan Cooper for her great efforts to help me finishing my study in University of Wollongong and AusAid for giving me the opportunity to study in Australia.

My gratitude also extends to the staffs of School of Information Technology and Computer Science for their help and encouragement. Finally I wish to thank to my family and friends who have helped to give me the strength and determination to finish this work.

Publication

The new material from this thesis has been included in a paper, J. Utami and J. Seberry. *On small defining sets for SBIBD(31,15,7)*, to be submitted for publication.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

We investigate the unique completion of a design. Given partial information, when can a design be uniquely completed? We study the minimal size of the partial information needed to uniquely complete a design. We also investigate the information contained in subsets of minimal defining sets.

The partial information required to construct a design closely relates to *secret sharing schemes* where a piece of confidential information is split into smaller, different shares of information. These smaller partial shares of information are distributed to different people who act together. We wish to know if the complete and valuable information can be reconstructed from their partial shares. We search for partial shares, which when combined, make it computationally infeasible to reconstruct the complete information.

1.2 Definitions

Let X be a finite set of points and let $\beta = \{B_i | i \in I\}$ be a family of subset B_i of X. The subset are called *blocks* and the pair (X, β) is called *a design* based on the set X. The order of a design (X, β) is $|X|$, the cardinality of X.

A design where all the blocks contain the same number of varieties, and the number of varieties is equal to the number of blocks is called *a symmetric block design.*

An incomplete block design is a design in which each block does not contain all the varieties.

A balanced incomplete block design or *BIBD* is a design where every pair of varieties occur together in exactly λ blocks. The parameters of a BIBD are v, b, r, k, λ which satisfy the following properties :

1. $bk = vr$

   This counts the total number of experimental unit or plots in two ways and hence gives a relationship that holds for all block designs.

2. $r(k-1) = \lambda(v-1)$

   This counts the number of times a given variety will occur with $(k-1)$ other varieties in each of r blocks and also the number of times each occurs with the other $(v-1)$ varieties in the λ blocks. This relationship holds for all $BIBD$.

$SBIBD$ stands for *symmetric balanced incomplete block design.* An $SBIBD$ is a $(v, b, r, k, \lambda) - configuration$ with the conditions that $v = b$ and $r = k$. An $SBIBD(v, k, \lambda)$ is also called a $(v, k, \lambda) - configuration$. **Statistical terminology** is used for the following terms and properties.

v is the number of varieties,
b is the number of blocks,
r is the replication number or number of occurrences of each variety,
k is the size of each block.
λ is the number of blocks in which each pair of varieties occurs.
λ is the number of varieties that occur in each pair of blocks.

One way to represent a design is by using *an incidence matrix.* A design (X, β) with v varieties and b blocks can be represented as $v \times b$ matrix, $A = [a_{ij}]$, such that :

$a_{ij} = 1$, if variety i belongs to block j and
$a_{ij} = 0$, if variety i does not belong to block j.

A $BIBD(v, b, r, k, \lambda)$ in the form of its incidence matrix is a $v \times b$ matrix with entries $0, 1$, and r ones per row, k ones per column, and the inner product of any pair of rows λ. The inner product of any pair of columns of an SBIBD is also λ.

In our representation, the rows are the varieties and the columns are the blocks so that element $(i, j) = 1$ means element i is in block j.

Example 1.2.1 The incidence matrix of a BIBD(9,12,4,3,1) :

$$\begin{array}{cccccccccccc}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0
\end{array}$$

Example 1.2.2 The incidence matrix of the first of Hussain's [5] three inequivalent $SBIBD(16, 6, 2)$:

$$\begin{array}{cccccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}$$

1.3 Minimal Defining Set

A minimal defining set, $\mathcal{D}$, is the size of the smallest set of rows which can be uniquely completed to give the required design, D. We call the solution vectors which can complete the design *candidate solutions* and the input vectors *starting rows*. The *Union*, $\mathcal{U}$, is used to describe the union of the candidate solutions and the starting rows, the term *hive*,H, is defined as $\mathcal{U}\backslash$ D, and we write q for the cardinality of the hive. For unique determination of the design, we would like q to be 0 but for good secret sharing properties we would like q to be as large as possible.

Consider the design 2-(7,3,1). (Refer to section 1.2 on this notation)

Now the 2-(7,3,1) design may be written as :

1.	1	1	1	0	0	0	0
2.	0	1	0	0	0	1	1
3.	0	0	1	1	0	1	0
4.	1	0	0	0	1	1	0
5.	1	0	0	1	0	0	1
6.	0	1	0	1	1	0	0
7.	0	0	1	0	1	0	1

We start with 3 blocks as an input :

1.	1	2	3	which have incidence matrix	1 0 0
2.	2	6	7		1 1 0
3.	3	4	6		1 0 1
					0 0 1
					0 0 0
					0 1 1
					0 1 0

In the remainder of this thesis we will always use the transpose of the incidence matrix.

Example 1.3.1 The transpose of incidence matrix of the 3 starting blocks is :

1.	1	1	1	0	0	0	0
2.	0	1	0	0	0	1	1
3.	0	0	1	1	0	1	0

To complete the design, we need to find all vectors with three ones per row which have inner product 1 with all the input vectors. The solution that we found is the unique completion of the design :

$$\begin{array}{llllllll} 4. & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 5. & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 6. & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 7. & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$$

Hence the rows $\{1,2,3\}$ give a defining set, but not necessarily a minimal defining set, for the 2-(7,3,1) design. In Example 1.3.1 the candidate solutions are rows $\{4,5,6,7\}$ and the starting rows are the rows $\{1,2,3\}$. Now, we wish to study the possible size of a minimal defining set.

Suppose one or more vectors are removed from the input set $\{1,2,3\}$ of Example 1.3.1. A complete search shows that no unique solution can be found. The following examples show that the set $\{1,2\}$ gave 9 candidate solutions which consist of 4 solutions from rows $\{4,5,6,7\}$, 1 solution from rows $\{1,2,3\}$ and 4 other solutions which together form the candidate solutions of $\{1,2,3\}$. The set $\{1,3\}$ gave 9 candidate solutions which are 4 solutions from rows $\{4,5,6,7\}$, 1 solution from rows $\{1,2,3\}$ and 4 other solutions, and finally the set $\{2,3\}$ gave 9 candidate solutions which are 4 solutions from rows $\{4,5,6,7\}$, 1 solution from rows $\{1,2,3\}$ and 4 other solutions.

Example 1.3.2 The following examples give the candidate solutions obtained for three 2-subsets of $\{1,2,3\}$.

When row 3 was removed from starting rows, we obtained the results of Table 1.1.

Table 1.1:

Starting Rows	Solutions from the Design	Other Solutions
1. 1 1 1 0 0 0 0	1 0 0 0 1 1 0	1 0 0 1 0 1 0
2. 0 1 0 0 0 1 1	1 0 0 1 0 0 1	1 0 0 0 1 0 1
	0 0 1 1 0 1 0	0 0 1 0 1 1 0
	0 1 0 1 1 0 0	0 0 1 1 0 0 1
	0 0 1 0 1 0 1	

When row 2 was removed from the starting rows, we obtained the results of Table 1.2.

Table 1.2:

Starting Rows	Solutions from the Design	Other Solutions
1. 1 1 1 0 0 0 0	1 0 0 0 1 1 0	1 0 0 1 1 0 0
3. 0 0 1 1 0 1 0	1 0 0 1 0 0 1	1 0 0 0 0 1 1
	0 1 0 0 0 1 1	0 1 0 0 1 1 0
	0 1 0 1 1 0 0	0 1 0 1 0 0 1
	0 0 1 0 1 0 1	

When row 1 was removed from starting rows, we obtained the results of Table 1.3.

Table 1.3:

Starting Rows	Solutions from the Design	Other Solutions
2. 0 1 0 0 0 1 1	1 0 0 0 1 1 0	0 0 0 1 1 0 1
3. 0 0 1 1 0 1 0	1 0 0 1 0 0 1	1 0 1 0 0 0 1
	1 1 1 0 0 0 0	1 1 0 1 0 0 0
	0 1 0 1 1 0 0	0 1 1 0 1 0 0
	0 0 1 0 1 0 1	

These small examples show the difficulty of using subsets of the original starting rows (3 inputs) to uniquely decide the remainder of the design. The subsets of the original starting rows each lead to 9 potential solutions.

The simple illustration above establishes that the minimal defining set for 2-(7,3,1) design is 3 vectors as no 2 vectors uniquely complete the design. The defining set of 3 blocks for the 2-(7,3,1) design has been mentioned by Sarvate and Seberry [10], Kunkle and Seberry [8] and Seberry [12] while the algorithm to construct a design by finding all vectors which have inner product λ with the defining set has been mentioned by Kunkle and Seberry [8].

Example 1.3.2 shows that the hive obtained from the starting rows $\{1,2\}$, $\{1,3\}$, and $\{2,3\}$ has cardinality 12.

Table 1.4:

Other Solutions
1. 1 0 0 1 0 1 0
2. 1 0 0 0 1 0 1
3. 0 0 1 0 1 1 0
4. 0 0 1 1 0 0 1
5. 1 0 0 1 1 0 0
6. 1 0 0 0 0 1 1
7. 0 1 0 0 1 1 0
8. 0 1 0 1 0 0 1
9. 0 0 0 1 1 0 1
10. 1 0 1 0 0 0 1
11. 1 1 0 1 0 0 0
12. 0 1 1 0 1 0 0

This Union $\mathcal{U}$ is the union of the "Solutions from the Design", "Other Solutions" from each of the tables 1.1, 1.2, and 1.3 and "The Starting Rows". The hive is $\mathcal{U} \setminus$ D which is the union given in table 1.4 of the union of "Other Solutions". In Fitina, Seberry and Chaudhry [2] the nest is used to discuss the completion of critical sets of *Latin Square*. We use hive as the arrays of $0, 1$ have some resemblance to a honeycomb.

Chapter 2

Secret Sharing Schemes

Suppose there is a piece of secret information, and we would like to distribute it to n persons. Each of these persons has his own share and has no knowledge of other people's shares. The people have to work together in order to reveal the secret. This scheme is called an $n - out - of - n$ *secret sharing scheme.* The person authorised as a distributor of the secret information is the *dealer* and each person who receives a partial secret information is a *shareholder or participant.*

In a perfect *n-out-of-n secret sharing scheme*, collaboration of any combination of fewer than the specified n shareholders, or any collusion of $n-1$ or fewer participants can not determine the secret.

From Stinson [14] an $n - out - of - n$ *secret sharing scheme* is a special case of a $(t, n) - threshold\ schemes$ or *Shamir Threshold Scheme* where $n = t$.
We now give the definition and an algorithm from Stinson [14] for the *Shamir Threshold Scheme* and a (n,n)-threshold scheme in $Z_m = \{0, 1, \cdots, m-1\}$ respectively.

Definition 2.0.1 *[14] Let t,n be positive integers, $t \leq n$. A (t,n)-***threshold scheme** *is a method of sharing a key K among a set of n participants (denoted by $\{P_1, \cdots, P-n\}$) in such a way that any t participants can compute the value of K, but no group of $t-1$ participants can do so.*

Algorithm 2.0.1 *[14] The Shamir (t,n)-threshold scheme in Z_p algorithm consists of two major steps :*

Initialization Step *and* **Share Distribution Step.**

Initialization Steps :

(1.) In this step the dealer selects n different and non-zero elements x_i from Z_p, where the value of x_i is public and $1 \leq i \leq n$. Then x_i is given to P_i. We require $n \leq p-1$ and p to be prime.

Share Distribution Step :

(2.) The dealer wants to share a key *K; so the dealer independently selects at random $t-1$ elements of Z_p, namely, $e_1, e_2, ..., e_{t-1}$.*

(3.) Then, for $1 \leq i \leq n$, the dealer calculates $y_i = e(x_i)$, where :

$$e(x) = K + \sum_{j=1}^{t-1} e_j x^j \ mod\ p$$

(4.) Finally, for $1 \leq i \leq n$, the dealer distributes the share y_i to P_i.

Algorithm 2.0.2 *[14] The (n,n)-threshold scheme in Z_m is given by the following steps :*

(1.) The dealer selects independently at random $n-1$ elements $y_1, ..., y_{n-1}$ of Zm.

(2.) The dealer calculates :

$$y_n = K - \sum_{i=1}^{n-1} y_i \ mod\ m.$$

(3.) Then, for $1 \leq i \leq n$, the dealer distributes the share y_i to P_i.

It is not required that $m \geq (n+1)$ and m is also not required to be prime. The n participants can calculate K from :

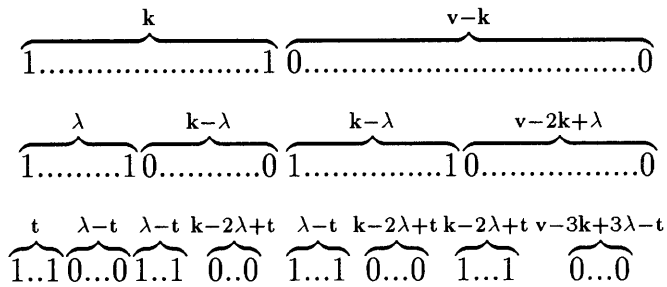
$$K = \sum_{i=1}^{n} y_i \ mod\ m.$$

Chapter 3

The Algorithms

We use three algorithms in this study. In particular, these algorithms are used to study the 2-(16,6,2) and 2-(31,15,7) designs.

Suppose we have n rows of the $(0,1)$ incidence matrix of an $SBIBD(v,k,\lambda)$, A. Without loss of generality, the first three rows may be written as :



With no rows chosen there are :

$$w_1 = \binom{v}{k} \tag{3.1}$$

possible choices for rows in A. If one row has been chosen there are :

$$w_2 = \binom{k}{\lambda}\binom{v-k}{k-\lambda} \tag{3.2}$$

possible choices for further rows in A. The formula for w_2 is obtained through the following steps :

From the first set of columns with length k we can generate $\binom{k}{\lambda}$ vectors. The second set of columns has length $v-k$ and we can generate $\binom{v-k}{k-\lambda}$ vectors. Hence, the total number of vectors is w_2.

Suppose two rows have been specified; then there are

$$w_3 = \sum_{t=0}^{\lambda} \binom{\lambda}{t} \binom{k-\lambda}{\lambda-t}^2 \binom{v-2k+\lambda}{k-2\lambda+t} \tag{3.3}$$

possible choices for further rows in A.

Formula (3.3) is obtained through the following steps.

First we want to generate vectors and length λ and weight t, where t runs from 0 to λ. This means that there are

$$\binom{\lambda}{t}$$

vectors. Then we look at the second set of columns which have length $k-\lambda$; the number of vectors is

$$\binom{k-\lambda}{\lambda-t}.$$

The third set of columns has the same length as the second set of columns; so the number of vectors is the same. This gives squaring of the term $\binom{k-\lambda}{\lambda-t}$. Finally, the last set of columns has length $v-2k+\lambda$ and we want to take $k-2\lambda+t$ ones from $v-2k+\lambda$. Thus, summing over all t, we obtain equation 3.3.

The following table is a summary of the numbers of rows generated by each algorithm.

Table 3.1:

(v,k,λ)	$\binom{v}{k}$	$\binom{k}{\lambda}\binom{v-k}{k-\lambda}$	$\sum_{t=0}^{\lambda}\binom{\lambda}{t}\binom{k-\lambda}{\lambda-t}^2\binom{v-2k+\lambda}{k-2\lambda+t}$
(7,3,1)	35	18	9
(13,4,1)	715	336	155
(16,6,2)	8008	3150	750
(19,9,4)	92378	31752	10626
(31,15,7)	300,540,195	82,818,450	22,458,249

Let us first consider an algorithm based on (3.1).

Algorithm 1:

1. Put the n initial vectors in a file. This uses nv bytes.
2. Create a file containing $\binom{v}{k}$ vectors with lengths v and weight k. This uses $v\binom{v}{k}$ bytes and requires $\mathcal{O}(v^{k+1})$ steps.
3. Test each of the w_1 vectors to see if its inner product with each of the n initial vectors is λ. Each test takes v $XORs$ (eXclusive OR) and $v-1$ additions : the total is $\mathcal{O}$ $(2nvw_1)$.
4. Output the vectors that pass the test in step 3.

Algorithm 2 (based on (3.2)):

1. Put the n initial vectors in a file. This uses nv bytes.

2a. Create a file containing $\binom{k}{\lambda}$ vectors with lengths k and weights λ. This uses $k\binom{k}{\lambda}$ bytes and requires $\mathcal{O}(k^{\lambda+1})$ steps.

2b. Create a file containing $\binom{v-k}{k-\lambda}$ vectors with lengths $v-k$ and weights $v-k$. This uses $(v-k)\binom{v-k}{k-\lambda}$ bytes and requires $\mathcal{O}((v-k)^{k-\lambda+1})$ steps.

3. Test each of the w_2 vectors formed by taking the first k coordinates from the file created at step 2a and the last $v-k$ coordinates from the file created at step 2b to see if its inner product with each of the n initial vectors is λ. Each test takes v $XORs$ and $v-1$ pluses; the total is $\mathcal{O}(2nvw_2)$.
4. Output the vectors that pass the test in step 3.

This algorithm uses a total storage of order $\mathcal{O}((v-k)^{k-\lambda+1})$ and a work factor of order $\mathcal{O}((v-k)^{k-\lambda+1})$.

We used Algorithm 2 and verified the Greenhill and Street [3] results that the size of the minimal defining sets for the three inequivalent 2-(16,6,2) designs of Husain [5] are 9, 7 and 7 respectively.

We now modify Algorithm 2 to an algorithm based on (3.3).

Algorithm 3 :

1. Put the n initial vectors in a file. This uses nv bytes.

2a. Create a file containing all 2^λ vectors of length λ and index them by their weight t. This uses $\lambda 2^\lambda$ bytes and requires $\mathcal{O}(\lambda 2^\lambda)$ steps.

2b. Create a file containing

$$\sum_{t=0}^{\lambda} \binom{k-\lambda}{\lambda-t}$$

vectors of length $k-\lambda$ and weight $\lambda - t$. This uses at most $(k-\lambda)2^{k-\lambda}$ bytes and requires $\mathcal{O}(k-\lambda)2^{k-\lambda})$ steps.

2c. Create a file containing

$$\sum_{t=0}^{\lambda} \binom{v-2k+\lambda}{k-2\lambda+t}$$

vectors of length $v-2k+\lambda$ and weight $k-2\lambda+t$. This uses $(v-2k+\lambda)$ $(2^{v-2k+\lambda}-2^{k-\lambda-1})$ bytes and requires $\mathcal{O}(v-2k+\lambda)2^{v-2k+\lambda})$ steps.

3. Test each of the w_3 vectors formed by taking the first λ coordinates with weight t from the file created at step 2a. As the second loop, take the $k-\lambda$ coordinates with weight $\lambda - t$ from the file created at step 2b. For the third loop, take $v-k$ coordinates with weight $\lambda - t$ from the file created at step 2b (it may be different from the second loop), and finally take $v-2k+\lambda$ coordinates with weight $k-2\lambda+t$ from the file created at step 2c to see if the inner product of the completely formed vector with each of the n starting vectors is λ. Each test takes v $XORs$ and $v-1$ pluses; the total is $\mathcal{O}(2nvw_3)$.

(4.) Output the vectors that pass the test in step 3.

This algorithm uses a total storage of $\mathcal{O}((v-2k+\lambda)2^{v-2k+\lambda})$. The number steps required is $\mathcal{O}((v-2k+\lambda)2^{v-2k+\lambda})$.

The following diagram is a summary of steps in Algorithm 3.

Suppose there are three files called A, B and C. File A has n_a vectors, file B has n_b vectors and file C has n_c vectors. The indexes of the vectors from each file are as follows.

File A provides coordinates $(1, \cdots, \lambda)$ of the candidate vectors.

File B provides coordinates $(\lambda + 1, \cdots, k)$ of the candidate vectors.

File B also provides coordinates $(k + 1, \cdots, 2k - \lambda)$ of the candidate vectors.

Finally, file C provides coordinates $(2k - \lambda + 1, \cdots, v)$ of the candidate vectors.

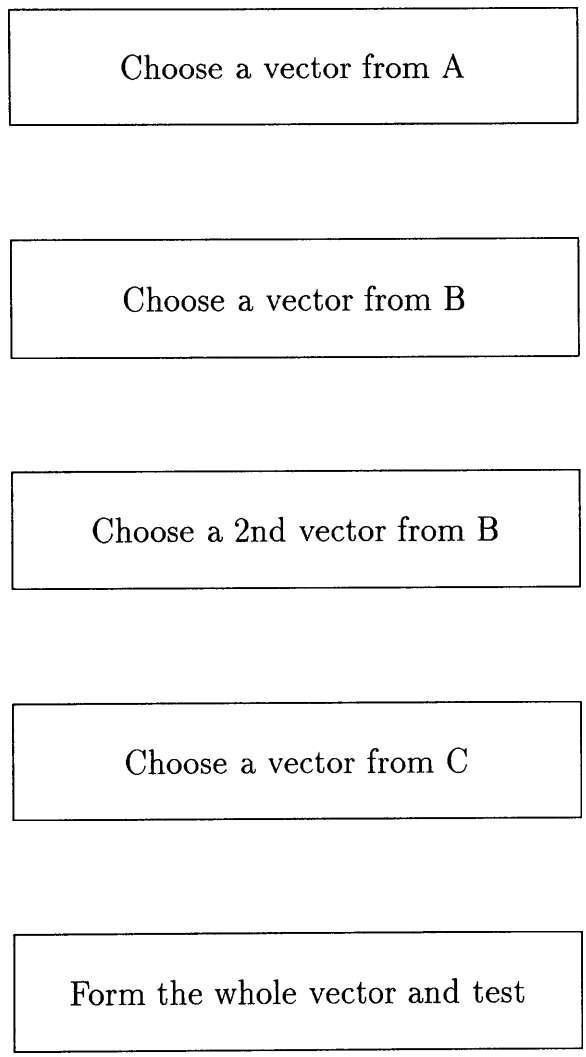


Figure 3.1:

In general, among the three algorithms, Algorithm 3 is the best. From Table 3.1, we can see that the difference in the number of test vectors produced by each algorithm

is significant when the size of the design become larger. This means that we can significantly reduce the total number of steps taken by using Algorithm 3.

Remark : Algorithm 3 requires

$$\sum_{s=0}^{t-1} \binom{t}{s}^4 \leq 2^{4t}$$

vectors of length 4t-1 to be created for an $SBIBD(4t-1, 2t-1, t-1)$ [8, 10, 12]

Proof:

Let us consider the following identity :

$$\sum_{t=0}^{\lambda} \binom{\lambda}{t} = \binom{\lambda}{0} + \cdots + \binom{\lambda}{\lambda}.$$

Now $(1+x)^\lambda = 1 + \binom{\lambda}{1}x + \cdots + \binom{\lambda}{\lambda}x^\lambda$. Hence, setting $x = 1$, we have

$$2^\lambda = 1 + \binom{\lambda}{1} + \cdots + \binom{\lambda}{\lambda}$$

Because all the x_i are nonnegative, it is clear that $(x_1 + x_2 + \cdots + x_s)^4 > {x_1}^4 + {x_2}^4 + \cdots + {x_s}^4$. Therefore

$$\sum_{s=0}^{t-1} \binom{t}{s}^4 \leq 2^{4t} = \left(\sum_{t=0}^{t-1} \binom{t}{s}\right)^4.$$

□

Chapter 4

The Design 2-(16,6,2)

The design 2-(16,6,2) is the second design that we want to investigate. From Husain [5], the design has three inequivalent sets given as Sets I, II, and III below, which are represented in the form of incidence matrices and alpha numerical symbols. This design is composed from 16 of the $\binom{16}{6} = 8008$ possible vectors with 6 ones in each row. We used Algorithm 1 to analyse this case.

4.1 The Completion Algorithm

The three inequivalent sets of 2-(16,6,2) designs are as follows:
Set I :

	Incidence Matrix	Variety/Block	
1.	1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	1 2 3 4 5 6	1
2.	1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0	1 2 7 8 9 A	2
3.	1 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0	1 3 7 B C D	3
4.	1 0 0 1 0 0 0 1 0 0 1 0 0 1 1 0	1 4 8 B E F	4
5.	1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1	1 5 9 C E G	5
6.	1 0 0 0 0 1 0 0 0 1 0 0 1 0 1 1	1 6 A D F G	6
7.	0 1 1 0 0 0 1 0 0 0 0 0 0 1 1 1	2 3 7 E F G	7
8.	0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 1	2 4 8 C D G	8
9.	0 1 0 0 1 0 0 0 1 0 1 0 1 0 1 0	2 5 9 B D F	9
10.	0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0	2 6 A B D E	10
11.	0 0 1 1 0 0 0 0 1 1 1 0 0 0 0 1	3 4 9 A B G	11
12.	0 0 1 0 1 0 0 1 0 1 0 1 0 0 1 0	3 5 8 A C F	12
13.	0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0	3 6 8 9 D E	13
14.	0 0 0 1 1 0 1 0 0 1 0 0 1 1 0 0	4 5 7 A D E	14
15.	0 0 0 1 0 1 1 0 1 0 0 1 0 0 1 0	4 6 7 9 C F	15
16.	0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 1	5 6 7 8 B G	16

The rows of the incidence matrix for Set I are called $DI1$, $DI2$,..., $DI16$, respectively.

Set II :

	Incidence Matrix	Variety/Block	
1.	1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	1 2 3 4 5 6	1
2.	1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0	1 2 7 8 9 A	2
3.	1 0 1 0 0 0 0 1 0 0 1 1 1 0 0 0	1 3 8 B C D	3
4.	1 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0	1 4 9 C E F	4
5.	1 0 0 0 1 0 0 0 0 1 1 0 0 1 0 1	1 5 A B E G	5
6.	1 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1	1 6 7 D F G	6
7.	0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0	2 3 7 B E F	7
8.	0 1 0 1 0 0 0 0 1 0 1 0 1 0 0 1	2 4 9 B D G	8
9.	0 1 0 0 1 0 0 0 0 1 0 1 1 0 1 0	2 5 A C D F	9
10.	0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 1	2 6 8 C D G	10
11.	0 0 1 1 0 0 1 0 0 1 0 1 0 0 0 1	3 4 7 A C G	11
12.	0 0 1 0 1 0 0 1 1 0 0 0 0 0 1 1	3 5 8 9 F G	12
13.	0 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0	3 6 9 A D E	13
14.	0 0 0 1 1 0 1 1 0 0 0 0 1 1 0 0	4 5 7 8 D E	14
15.	0 0 0 1 0 1 0 1 0 1 1 0 0 0 1 0	4 6 8 A B F	15
16.	0 0 0 0 1 1 1 0 1 0 1 1 0 0 0 0	5 6 7 9 B C	16

The rows of the incidence matrix for Set II are called $DII1$, $DII2$, ..., $DII16$, respectively.

Set III :

	Incidence Matrix	Variety/Block	
1.	1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	1 2 3 4 5 6	1
2.	1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0	1 2 7 8 9 A	2
3.	1 0 1 0 0 0 0 1 0 0 1 1 1 0 0 0	1 3 8 B C D	3
4.	1 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0	1 4 9 B D F	4
5.	1 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1	1 5 A C F G	5
6.	1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1	1 6 7 D E G	6
7.	0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 1	2 3 7 B F G	7
8.	0 1 0 1 0 0 0 0 1 0 0 1 1 0 0 1	2 4 9 C D G	8
9.	0 1 0 0 1 0 0 0 0 1 1 0 1 1 0 0	2 5 A B D E	9
10.	0 1 0 0 0 1 0 1 0 0 0 1 0 1 1 0	2 6 8 C E F	10
11.	0 0 1 1 0 0 1 0 0 1 0 1 0 1 0 0	3 4 7 A C E	11
12.	0 0 1 0 1 0 0 1 1 0 0 0 0 1 0 1	3 5 8 9 E G	12
13.	0 0 1 0 0 1 0 0 1 1 0 0 1 0 1 0	3 6 9 A D F	13
14.	0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0	4 5 7 8 D F	14
15.	0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 1	4 6 8 A B G	15
16.	0 0 0 0 1 1 1 0 1 0 1 1 0 0 0 0	5 6 7 9 B C	16

The rows of incidence matrix for Set III are called $DIII1$, $DIII2$, ...,$DIII16$, respectively.

The algorithm we used to find the solutions for each design is different from the algorithm used by Greenhill and Street [3]. We use the term *inner product*, λ, to indicate the number of pairs of varieties that occur in the same block of a design as mentioned by Kunkle and Seberry [8]. Thus, if we consider a symmetric 2-(v, k, λ) design, any pair of rows, where each row has maximum k ones, has to have inner product λ with each other row. Our algorithm has time complexity $\mathcal{O}(k^{\lambda+1})$ which we believe is less than that of Greenhill and Street [3], though they do not compute complexity explicitly.

In [3], each block was obtained by considering all the block design properties. We obtained the same results as Greenhill and Street but our algorithm is elegant to program. For small design size, such as for the design 2-(7,3,1), the elegance and complexity is not really a problem; however, for larger designs such as 2-(16,6,2) or 2-(31,15,7), the computational time and the storage required by the algorithm severely limits the size of design which can be analysed.

From Greenhill and Street [3], the size of the minimal defining sets of sets I, II, III are 9, 7, and 7 respectively. The following are examples of the completion of the three designs with given minimal defining sets.

Example 4.1.1 In Tables 3.1, 3.2 and 3.3, if we start with the "Starting Rows" given our algorithm shows that the design is completed by the rows given in "Solutions from Set" I, II, III, respectively.

Table 4.1:

Starting Rows	Solutions from Set I
1. $DI1$	1. $DI6$
2. $DI2$	2. $DI8$
3. $DI3$	3. $DI10$
4. $DI4$	4. $DI11$
5. $DI5$	5. $DI12$
6. $DI7$	6. $DI14$
7. $DI9$	7. $DI15$
8. $DI13$	
9. $DI16$	

Table 4.2:

Starting Rows	Solutions from Set II
1. $DII1$	1. $DII15$
2. $DII2$	2. $DII5$
3. $DII3$	3. $DII6$
4. $DII4$	4. $DII7$
5. $DII10$	5. $DII11$
6. $DII14$	6. $DII8$
7. $DII16$	7. $DII9$
	8. $DII12$
	9. $DII13$

Table 4.3:

Starting Rows	Solutions from Set III
1. $DIII1$	1. $DIII15$
2. $DIII2$	2. $DIII6$
3. $DIII3$	3. $DIII11$
4. $DIII4$	4. $DIII14$
5. $DIII5$	5. $DIII8$
6. $DIII7$	6. $DIII9$
7. $DIII16$	7. $DIII10$
	8. $DIII12$
	9. $DIII13$

This confirms the results of Greenhill and Street [3].

4.2 The Influence of Row Removal

As we wish to discover the usefulness of designs for secret sharing, we would like to know more of the candidate solutions, C, and the hive, H, for each design. We remove, systematically, one or more vectors from the set of starting rows. This enables us to see if we truly have an $n-out-of-n$ *scheme*. We systematically remove one vector at a time from Sets I, II and III. The complete results are given in the Tables and Examples below.

Example 4.2.1 We consider the first example, Example 4.1.1. We observe the effect of systematically removing rows from a minimal defining set to find the number of candidate solutions for each subset of the minimal defining set. We use the names NonDI, NonDII, Non$DIII$ for vectors not from sets I, II, and III, respectively.

When row 2 was removed from the starting rows of set I, we obtained Table 4.4.

Table 4.4:

Starting Rows	Solutions from Set I	Other Solutions
1. $DI1$	1. $DI15$	1.Non$DI1$
2. $DI3$	2. $DI6$	2.Non$DI2$
3. $DI4$	3. $DI11$	3.Non$DI3$
4. $DI5$	4. $DI14$	4.Non$DI4$
5. $DI7$	5. $DI8$	
6. $DI9$	6. $DI10$	
7. $DI13$	7. $DI12$	
8. $DI16$	8. $DI2$	

This means that all the other rows from set I were generated (as expected) plus another four; in total, 12 rows with 6 ones per row had inner product 2 with each of the starting rows.

The Table 4.5 gives the number of candidate solutions produced when we removed each starting row in turn from the set of set I.

Table 4.5:

Starting Row Removed	Number of Candidate Solutions from Design I
$DI1$	11
$DI2$	12
$DI3$	12
$DI4$	12
$DI5$	12
$DI7$	16
$DI9$	16
$DI13$	16
$DI16$	16

The table above gives, for each possible $(n-1)-out-of-n$ subset, the number of rows satisfying the conditions of weight 6 and inner product 2. Since we know 8 rows are missing from each design, a person seeking to cheat must guess 8 out of the candidate solutions from Design I. In each case, the probability of guessing correctly is $\binom{|C|}{8}$. This shows that probability of guessing the *correct solutions* is still too high.

Now, let see us the effect of row removal on the number of candidate solutions, and the probability of guessing the correct solutions for each of designs II and III.

When row 2 was removed from the starting rows of set II, we obtained Table 4.6.

Table 4.6:

Starting Rows	Solutions from Set II	Other Solutions
1. $DII1$	1. $DII15$	1.Non$DII1$
2. $DII3$	2. $DII5$	2.Non$DII2$
3. $DII4$	3. $DII6$	3.Non$DII3$
4. $DII10$	4. $DII7$	4.Non$DII4$
5. $DII14$	5. $DII11$	5.Non$DII5$
6. $DII16$	6. $DII8$	6.Non$DII6$
	7. $DII9$	7.Non$DII7$
	8. $DII12$	8.Non$DII8$
	9. $DII13$	
	10.$DII2$	

Hence, a cheater would need to guess 10 out of 18, which has probability $1/\binom{18}{10}$ which is unacceptably high.

When row 2 was removed from the starting rows of set III, we obtained Table 4.7.

Table 4.7:

Starting Rows	Solutions from Set III	Other Solutions
1. $DIII1$	1. $DIII15$	1.Non$DIII1$
2. $DIII3$	2. $DIII6$	2.Non$DIII2$
3. $DIII4$	3. $DIII8$	3.Non$DIII3$
4. $DIII5$	4. $DIII9$	4.Non$DIII4$
5. $DIII7$	5. $DIII11$	5.Non$DIII5$
6. $DIII16$	6. $DIII14$	6.Non$DIII6$
	7. $DIII10$	7.Non$DIII7$
	8. $DIII12$	8.Non$DIII8$
	9. $DIII13$	9.Non$DIII9$
	10.$DIII2$	10.Non$DIII10$

In this case a cheater would need to guess 10 out of 20 candidate solutions, this means that the probability of guessing the correct solutions is $1/\binom{20}{10}$

We ran a computer program to find the candidate solutions in each case, in each Set, removing one row from the set of starting rows at a time; we summarise the results to obtain the hive of each Set.

A summary of the size of the hive for sets I, II and III is given in Table 4.8.

Table 4.8:

	Set I	Set II	Set III
Size of the hive :	51	65	80

The hive was obtained by using formula $\mathcal{U} \setminus$ D or by adding all the candidate solutions which are not from the design itself in each case in each Set.

By removing one vector from each set of starting rows, we found more candidate solutions including those from the design itself. We see that the hives, H, of the second and the third design are larger than that of the first design. We conjecture that smaller sets of starting rows in the minimal defining set give more candidate solutions; we conjecture that smaller minimal defining sets give secret sharing schemes with less chance of cheating. We also would like to see the number of candidate solutions from sets I, II, and III if we remove more than one row at a time. The following examples summarise the solutions obtained when two or three vectors were removed from the starting rows.

Example 4.2.2 Consider the removal of two or three vectors from the starting rows of set I, II, and III. Some sample results are as follows :

When rows 2 and 3 were removed from the starting rows of set I, we obtained Table 4.9.

Table 4.9:

Starting Rows	Solutions from Set I	Other Solutions
1. $DI1$	1. $DI15$	1.Non$DI1$
2. $DI4$	2. $DI6$	2.Non$DI2$
3. $DI5$	3. $DI11$	3.Non$DI3$
4. $DI7$	4. $DI14$	4.Non$DI4$
5. $DI9$	5. $DI8$	5.Non$DI5$
6. $DI13$	6. $DI10$	6. Non$DI6$
7. $DI16$	7. $DI12$	7. Non$DI7$
	8. $DI2$	8. Non$DI8$
	9. $DI3$	9. Non$DI9$
		10. Non$DI10$
		11. Non$DI11$
		12. Non$DI12$

When rows 2, 3, and 4 were removed from starting rows of set I, we obtained Table 4.10.

Table 4.10:

Starting Rows	Solutions from Set I	Other Solutions
1. $DI1$	1. $DI15$	1.Non$DI1$
2. $DI5$	2. $DI6$	2.Non$DI2$
3. $DI7$	3. $DI11$	3.Non$DI3$
4. $DI9$	4. $DI14$	4.Non$DI4$
5. $DI13$	5. $DI8$	5.Non$DI5$
6. $DI16$	6. $DI10$	6. Non$DI6$
	7. $DI12$	7. Non$DI7$
	8. $DI2$	8. Non$DI8$
	9. $DI3$	9. Non$DI9$
	10.$DI4$	10. Non$DI10$
		11. Non$DI11$
		12. Non$DI12$

Starting Rows(Cont.)	Solutions from Set I	Other Solutions
		13. NonDI13
		14. NonDI14
		15 .NonDI15
		16.NonDI16
		17.NonDI17
		18.NonDI18
		19. NonDI19
		20. NonDI20
		21. NonDI21
		22. NonDI22
		23. NonDI23
		24. NonDI24

When rows 2, and 3 were removed from starting rows of set II, we obtained Table 4.11.

Table 4.11:

Starting Rows	Solutions from Set II	Other Solutions
1. DII1	1. DII15	1.NonDII1
2. DII4	2. DII5	2.NonDII2
3. DII10	3. DII6	3.NonDII3
4. DII14	4. DII7	4.NonDII4
5. DII16	5. DII11	5.NonDII5
	6. DII8	6.NonDII6
	7. DII9	7.NonDII7
	8. DII12	8.NonDII8
	9. DII13	9. NonDII9
	10.DII2	10. NonDII10
	11.DII3	11. NonDII11
		12. NonDII12

Starting Rows (Cont.)	Solutions from Set II	Other Solutions
		13.NonDII13
		14.NonDII14
		15.NonDII15
		16. NonDII16
		17. NonDII17
		18. NonDII18
		19. NonDII19
		20. NonDII20
		21.NonDII21
		22.NonDII22
		23.NonDII23
		24. NonDII24
		25. NonDII25
		26. NonDII26
		27. NonDII27
		28. NonDII28
		29.NonDII29
		30.NonDII30
		31.NonDII31
		32. NonDII32
		33. NonDII33
		34. NonDII34
		35. NonDII35
		36. NonDII36
		37.NonDII37
		38.NonDII38
		39.NonDII39
		40. NonDII40

When rows 2 and 3 were removed from the starting rows of set III, we obtained Table 4.12.

Table 4.12:

Starting Rows	Solutions from Set III	Other Solutions
1. $DIII1$	1. $DIII15$	1.Non$DIII1$
2. $DIII4$	2. $DIII10$	2.Non$DIII2$
3. $DIII5$	3. $DIII6$	3.Non$DIII3$
4. $DIII7$	4. $DIII14$	4.Non$DIII4$
5. $DIII16$	5. $DIII11$	5.Non$DIII5$
	6. $DIII8$	6.Non$DIII6$
	7. $DIII9$	7.Non$DIII7$
	8. $DIII12$	8.Non$DIII8$
	9. $DIII13$	9. Non$DIII9$
	10.$DIII2$	10. Non$DIII10$
	11.$DIII3$	11. Non$DIII11$
		12. Non$DIII12$
		13.Non$DIII13$
		14.Non$DIII14$
		15.Non$DIII15$
		16. Non$DIII16$
		17. Non$DIII17$
		18. Non$DIII18$
		19. Non$DIII19$
		20. Non$DIII20$
		21.Non$DIII21$
		22.Non$DIII22$
		23.Non$DIII23$
		24. Non$DIII24$

Starting Rows(Cont.)	Solutions from Set III	Other Solutions
		25. Non$DIII$25
		26. Non$DIII$26
		27. Non$DIII$27
		28. Non$DIII$28
		29.Non$DIII$29
		30.Non$DIII$30
		31.Non$DIII$31
		32. Non$DIII$32
		33. Non$DIII$33
		34. Non$DIII$34
		35. Non$DIII$35
		36. Non$DIII$36
		37.Non$DIII$37
		38.Non$DIII$38
		39. Non$DIII$39
		40. Non$DIII$40
		41. Non$DIII$41
		42. Non$DIII$42
		43. Non$DIII$43

The candidate solutions when rows 2, 3, and 4 were removed from starting rows in set II comprised 154 vectors which consist of 12 solutions from set II and 142 other solutions while the number of candidate solutions when rows 2, 3, and 4 were removed from starting rows of set III comprised 152 vectors which consist of 12 solutions from set III and 140 other solutions.

The candidate solutions of each design became larger when two or more rows were removed from the starting rows and this result leads to a larger hive, H.

We are interested in investigating the probability of guessing the solutions from the set of starting rows of set III when l ($l = 1, 2, 3, ..., \mathcal{D}$-1) rows are removed from the starting rows. We chose set III because, from the summary of section 3.1 , this set gave a larger size hive than the other two. The results of our computer search are summarised in Table 4.13.

From Table 4.13, we see that the probability of guessing the correct solutions from the design becomes smaller as the number of rows removed increases. However, in general

Table 4.13: Summary of Set III

l rows removed from D for set III	Number of Solutions	Probability
1	20	$1/\binom{20}{10}$
2	54	$1/\binom{54}{11}$
3	152	$1/\binom{152}{12}$
4	435	$1/\binom{435}{13}$
5	1195	$1/\binom{1195}{14}$
6	3150	$1/\binom{3150}{15}$

the probability of guessing the *correct solution* from the three designs of size 16 is still high, especially for design I. For design I, it is suggested that the nine rows should not be made into a $9-out-of-9$ scheme but rather into smaller group for example, as sets of 3 shares of 3 rows making a $3-out-of-3$ scheme. Then the probability of guessing the *correct solutions* becomes smaller; an opponent has less chance to cheat. However, the results for the 2-(16,6,2) design show that these designs are too small for practical secret sharing schemes. We suggest that a practical scheme should have probability of guessing correctly $< 1/10^{40}$

Chapter 5

The Design 2-(31,15,7)

The 2-(31,15,7) design is the third design that we want to investigate. This design has $\binom{31}{15} = 300,540,195$ possible test vectors which could be part of a minimal defining set. Each row has fifteen ones and any pair of rows has inner product precisely λ.

We used two types of data as starting rows. The two types of data are the *Paley Construction* and the *MarshallHall Construction* [4], where each group of starting rows contains 15 rows.

5.1 Paley Construction

The Paley Construction is based on the difference set of the residues modulo a prime power congruent to 3 (mod 4). Therefore, for $v = 31$, we select the difference set, D, $D = \{y : y = x^2 (mod\ v), x \neq 0\}$. The elements 0 to $v - 1$ are indices of the first row of the incidence matrix of the $SBIBD$, $A = a_{1j}$, where :

$$a_{1j} = \begin{cases} 1 & j \in \text{D} \\ 0 & \text{otherwise} \end{cases}$$

The remaining rows are completed by using the following formula :

$$a_{ij} = a_{i-1,j-1}.$$

15 $(= \frac{1}{2}(v-1))$ starting rows of the *Paley Construction* are as follow :

1.	2	3	5	6	8	9	A	B	F	H	J	K	L	Q	T
2.	3	4	6	7	9	A	B	C	G	I	K	L	M	R	U
3.	1	5	6	8	9	B	C	D	E	I	K	M	N	O	T
4.	2	6	7	9	A	C	D	E	F	J	L	N	O	P	U
5.	1	4	8	9	B	C	E	F	G	H	L	N	P	Q	R
6.	2	5	9	A	C	D	F	G	H	I	M	O	Q	R	S
7.	3	6	A	B	D	E	G	H	I	J	N	P	R	S	T
8.	4	7	B	C	E	F	H	I	J	K	O	Q	S	T	U
9.	1	2	3	8	B	F	G	I	J	L	M	N	O	S	U
10.	1	3	4	5	A	D	H	I	K	L	N	O	P	Q	U
11.	1	3	5	6	7	C	F	J	K	M	N	P	Q	R	S
12.	2	4	6	7	8	D	G	K	L	N	O	Q	R	S	T
13.	3	5	7	8	9	E	H	L	M	O	P	R	S	T	U
14.	1	2	3	4	8	A	C	D	E	J	M	Q	R	T	U
15.	1	2	4	5	6	7	B	D	F	G	H	M	P	T	U

From our computer search, we obtained 16 candidate solutions which uniquely complete the design. Seberry [12] conjectured that for p an odd prime or prime power congruent to 3 mod 4, a particular set of $\frac{1}{2}(p-1)$ rows can be used to uniquely complete an $SBIBD(p, \frac{1}{2}(p-1), \frac{1}{4}(p-3))$. From the result above, we see that these 15 starting rows for the Paley Construction do uniquely define an $SBIBD(31, 15, 7)$ design where $p = 31$ is congruent to 3 mod 4 and $\frac{1}{2}(p-1) = 15$. This means that a conjecture stated in Kunkle and Seberry [8] is also true for p = 31 as well as for the previously known p = 7, 11, 19, 23, 27, 43, 47, 59, and 67. The completion of this design uses the 15 starting rows to construct the $\frac{1}{2}(p+1)$ *or* 16 related rows which have inner product $\frac{1}{2}(p-3)$ with the starting rows. Sarvate and Seberry [10] give a similar conjecture for $p \equiv 1 \ (mod\ 4)$.

By computation, every 14-subset of 15-set is also a defining set for the $Paley(31, 15, 7)$ design. We used Algorithm 3 on all 13-subsets of the 15 rows of the $BIBD(15, 30, 14, 7, 6)$ which is derived from the Paley $SBIBD(31, 15, 7)$. We found 13 of the 13-subsets gave minimal defining sets. We tested all $\binom{15}{12}$ 12-subsets of the 15 rows above and found none of then completed uniquely to the design.

We list the 13 sets of the 13-subsets which gave minimal defining sets in Table 5.1.

Table 5.1:

No.	13-subsets of Starting Rows
1.	1 2 3 4 5 6 7 8 9 10 12 13 15
2.	1 2 3 4 5 6 7 8 10 11 12 14 15
3.	1 2 3 4 5 6 7 8 11 12 13 14 15
4.	1 2 3 4 5 6 8 9 10 11 12 13 14
5.	1 2 3 4 5 7 8 9 10 12 13 14 15
6.	1 2 3 4 6 7 9 10 11 12 13 14 15
7.	1 2 3 5 6 7 8 9 10 11 13 14 15
8.	1 3 4 5 6 7 8 9 10 11 12 14 15
9.	1 3 4 5 6 7 8 9 10 11 12 13 15
10.	1 2 4 5 6 8 9 10 11 12 13 14 15
11.	1 2 4 5 6 7 8 9 10 11 13 14 15
12.	2 3 4 5 6 7 8 9 10 11 12 13 14
13.	2 3 4 5 6 7 9 10 11 12 13 14 15

Conjecture 1 *The minimal defining set of the Paley* $SBIBD(31, 15, 7)$ *has* 13 *complete blocks.*

Tables 5.2 and 5.2 present a summary of the results of our computer search for the sizes of the set of candidate solutions, the probabilities of guessing the correct solutions, and the sizes of the hive in Table 5.2 when each different row is removed from the 13 *starting rows* one at a time.

The size of hive of 2-(31,15,7) with Paley Construction is 159 which was obtained from the formula : $\mathcal{U} \setminus$ D. Table 5.2 shows that the probability of guessing the *correct solutions* from the *candidate solutions* shown above is unacceptably high. We conjecture that the probability of guessing the correct solutions becomes lower as we remove more rows from the set of starting rows. Table 5.3 is a summary of the computational results when two or more rows were removed from the set of starting rows.

The results from Table 5.1, Table 5.2 and Table 5.3 enable us to define the 13 rows that give a minimal defining set for the Paley Construction of the 2-(31,15,7) design.

Table 5.2: Summary of Paley Construction when one row was removed

	Starting Row Removed	Number of Candidate Solutions	Probability
1.	1st row	28	$1/\binom{28}{19}$
2.	2nd row	29	$1/\binom{29}{19}$
3.	3rd row	32	$1/\binom{32}{19}$
4.	4th row	30	$1/\binom{30}{19}$
5.	5th row	31	$1/\binom{31}{19}$
6.	6th row	28	$1/\binom{28}{19}$
7.	7th row	33	$1/\binom{33}{19}$
8.	8th row	32	$1/\binom{32}{19}$
9.	9th row	34	$1/\binom{34}{19}$
10.	10th row	32	$1/\binom{32}{19}$
11.	11th row	32	$1/\binom{32}{19}$
12.	12th row	32	$1/\binom{32}{19}$
13.	13th row	33	$1/\binom{33}{19}$

Table 5.3: Summary of Paley Construction

Number of Rows Removed	Number of Candidate Solutions	Probability
1	33	$1/\binom{33}{12}$
2	101	$1/\binom{101}{11}$
3	383	$1/\binom{383}{10}$
4	1620	$1/\binom{1620}{9}$
5	5838	$1/\binom{583}{8}$
6	8624	$1/\binom{8624}{7}$

5.2 Marshall Hall Construction

The second set of input data that we used is the *Marshall Hall Construction* . The Hall construction is also based on a difference set, M. The incidence matrix of the Hall difference set is formed precisely as the incidence matrix of the Paley difference set with $M = \{2, 3, 4, 5, 7, 9, 14, 16, 17, 18, 25, 28, 30, 31\}$ from Hall [4].

As the case of *Paley Construction*, the 15 rows of vectors as starting rows from the *Marshall Hall Construction* are :

1.	2	3	4	5	7	9	D	G	H	I	O	P	S	U	V
2.	1	3	4	5	6	8	A	E	H	I	J	P	Q	T	V
3.	1	2	4	5	6	7	9	B	F	I	J	K	Q	R	U
4.	2	3	5	6	7	8	A	C	G	J	K	L	R	S	V
5.	1	3	4	6	7	8	9	B	D	H	K	L	M	S	T
6.	3	5	6	8	9	A	B	D	F	J	M	N	O	U	V
7.	1	2	5	7	8	A	B	C	D	F	H	L	O	P	Q
8.	5	6	9	B	C	E	F	G	H	J	L	P	S	T	U
9.	1	2	8	9	C	E	F	H	I	J	K	M	O	S	V
10.	1	2	3	9	A	D	F	G	I	J	K	L	N	P	T
11.	2	3	4	A	B	E	G	H	J	K	L	M	O	Q	U
12.	1	5	8	9	A	G	H	K	M	N	P	Q	R	S	U
13.	2	6	9	A	B	H	I	L	N	O	Q	R	S	T	V
14.	1	3	5	9	C	D	E	K	L	O	Q	R	T	U	V
15.	1	2	3	5	7	B	E	F	G	M	N	Q	S	T	V

Through our computer program search, we found 269 candidate solutions. This means that we have to add more vectors to the starting rows in order to reduce the number of candidate solutions. We added one row at a time to investigate the candidate solutions, finally we found that with 19 starting rows the design was uniquely completable. We found the following 19 rows yield a defining set :

1.	2	3	4	5	7	9	D	G	H	I	O	P	S	U	V
2.	1	3	4	5	6	8	A	E	H	I	J	P	Q	T	V
3.	1	2	4	5	6	7	9	B	F	I	J	K	Q	R	U
4.	2	3	5	6	7	8	A	C	G	J	K	L	R	S	V
5.	1	3	4	6	7	8	9	B	D	H	K	L	M	S	T
6.	3	5	6	8	9	A	B	D	F	J	M	N	O	U	V
7.	1	2	5	7	8	A	B	C	D	F	H	L	O	P	Q
8.	5	6	9	B	C	E	F	G	H	J	L	P	S	T	U
9.	1	2	8	9	C	E	F	H	I	J	K	M	O	S	V
10.	1	2	3	9	A	D	F	G	I	J	K	L	N	P	T
11.	2	3	4	A	B	E	G	H	J	K	L	M	O	Q	U
12.	1	5	8	9	A	G	H	K	M	N	P	Q	R	S	U
13.	2	6	9	A	B	H	I	L	N	O	Q	R	S	T	V
14.	1	3	5	9	C	D	E	K	L	O	Q	R	T	U	V
15.	1	2	3	5	7	B	E	F	G	M	N	Q	S	T	V
16.	3	5	7	B	C	H	I	J	K	M	N	O	P	R	T
17.	1	2	4	5	6	D	E	J	L	M	N	O	P	R	S
18.	1	4	7	8	F	G	H	J	L	N	O	R	T	U	V
19.	7	8	9	B	D	E	G	I	J	L	M	P	Q	R	V

The candidate solutions are :

1.	1	6	7	A	C	D	G	I	J	M	O	Q	S	T	U
2.	3	6	7	8	E	F	I	K	L	N	O	P	Q	S	U
3.	4	5	6	C	D	F	G	H	I	K	L	M	N	Q	V
4.	2	4	8	B	C	D	J	K	N	P	Q	S	T	U	V
5.	1	2	3	6	8	B	C	D	E	G	H	I	N	R	U
6.	1	4	6	7	9	A	B	C	E	G	K	N	O	P	V
7.	2	4	5	7	8	9	A	C	E	I	L	M	N	T	U
8.	3	4	7	9	A	C	D	E	F	H	J	N	Q	R	S
9.	2	3	4	6	8	9	C	F	G	M	O	P	Q	R	T
10.	4	5	8	A	B	D	E	F	G	I	K	O	R	S	T
11.	1	3	4	A	B	C	F	I	L	M	P	R	S	U	V
12.	2	6	7	A	D	E	F	H	K	M	P	R	T	U	V

The 19 rows of starting rows the from the *Marshall Hall Construction* gave a defining set for a 2-(31,15,7) design. Now, we would like to know the influence of row removal

from the set of starting rows on the number of candidate solutions. The table below shows the candidate solutions and the probabilities of guessing the correct solutions when one row is removed from the set of 19 starting rows:

Table 5.4: Summary of Marshall Hall Construction with one row removed

	Starting row removed	Size of Candidate Solutions	Probability
1.	1st	17	$1/\binom{17}{13}$
2.	2nd	21	$1/\binom{21}{13}$
3.	3rd	18	$1/\binom{18}{13}$
4.	4th	17	$1/\binom{17}{13}$
5.	5th	17	$1/\binom{17}{13}$
6.	6th	29	$1/\binom{29}{13}$
7.	7th	21	$1/\binom{21}{13}$
8.	8th	21	$1/\binom{21}{13}$
9.	9th	21	$1/\binom{21}{13}$
10.	10th	17	$1/\binom{17}{13}$
11.	11th	21	$1/\binom{21}{13}$
12.	12th	17	$1/\binom{17}{13}$
13.	13th	17	$1/\binom{17}{13}$
14.	14th	21	$1/\binom{21}{13}$
15.	15th	20	$1/\binom{20}{13}$
16.	16th	29	$1/\binom{29}{13}$
17.	17th	29	$1/\binom{29}{13}$
18.	18th	17	$1/\binom{17}{13}$
19.	19th	17	$1/\binom{17}{13}$

As we mention in the previous chapter, for a good secret sharing schemes, the size of the hive is important; we would like to have a large size hive. The hive for the 2-(31,15,7) design with the *Marshall Hall Construction*, when one row was removed from the set of 19 starting rows, is 153 which is smaller than the size of the hive for the Paley Construction when one row was removed from the set of 13 starting rows. The probability of guessing the correct solution is still high. The Table 5.5 shows the number of candidate solutions and the probabilities of guessing the correct solutions when more rows were removed from the starting rows.

Table 5.5: Summary of Marshall Hall Construction

Number of Rows Removed	Number of Candidate Solutions	Probability
1	17	$1/\binom{17}{13}$
2	46	$1/\binom{46}{14}$
3	99	$1/\binom{99}{15}$
4	269	$1/\binom{269}{16}$
5	563	$1/\binom{563}{17}$
6	1140	$1/\binom{1140}{18}$

From the results shown in Table 5.3 and Table B.2 we have a defining set for the 2-(31,15,7) design starting with the *Marshall Hall* data with 19 blocks. The 19 blocks of the *Marshall Hall Construction* are a *defining set* for a 2-(31,15,7) design.

Conjecture 2 *One minimal defining set of the Marshall Hall* $SBIBD(31, 15, 7)$ *contains* 19 *blocks.*

The 2-(31,15,7) design has 13 rows as a minimal defining set for the *Paley Construction* and 19 rows as a defining set for the *Marshall Hall Construction.* From Table 5.3 and Table 5.5, it is clear that the 13 rows of the minimal defining set of the Paley Construction gave a smaller probability of guessing the complete design when further rows were removed compared with the probability of guessing given by the 19 rows of the defining set of the *Marshall Hall Construction.* For good secret sharing schemes, the *Paley Construction* with 13 starting rows is better than the *Marshall Hall Construction* with 19 rows. In chapter 3 we conjectured that a smaller number of starting rows in the minimal defining set gives more candidate solutions and a lower probability of guessing the correct solution. This means that with smaller minimal defining sets, the secret sharing schemes will give the cheater(s) less chance to cheat.

Chapter 6

SBIBD in N-out-of-N Secret Sharing Scheme

Suppose there are n *initial rows* in the form of binary numbers. These n *initial rows* are then distributed to n *participants* or n *shareholders* where each participant has no knowledge about the design and the shares of the other participants. Then we have an *n-out-of-n secret sharing scheme;* working together these n *participants* can reconstruct the whole design.

What will happen if one or more participants wants to reconstruct the design alone without cooperating with other legitimate participants ?

Examples in the previous chapters have shown us that, if the number of rows used is less than the size of the minimal defining set, then a coalition of shareholders working together will find more candidate solutions than just the design. Such a coalition has to guess from the solutions which candidate solutions are the *correct solutions*. The more candidate solutions presented, the greater the difficulty of guessing the *correct solution*. It is clear that, with a small probability of guessing the correct solutions, a cheater has less chance to cheat. Ideally, for a secure secret sharing scheme, we would like to have the probability of guessing the correct solutions as small as possible.

To be precise: suppose we have $m < n$ shares or starting rows, which when tested for candidate solutions with correct weight and correct inner product with each of the starting rows, yields c candidate solutins, $c + n > v$. Then a cheating coalition of m shareholders has to guess $v - m$ out of c potensial solutions. There are $\binom{c}{(v-m)}$ ways to do this and so the probability of guessing the correct solutions is $1/\binom{c}{(v-m)}$.

We illustrate our proposed secret sharing scheme using the three $SBIBD(16, 6, 2)$.

We give one row of the minimal defining set to each participant. This means there are 9 participants for a scheme based on Set I and 7 participants for schemes based on

Sets II and III. We use n for the number of participants.

If n participants collaborate they can together recreate a unique instance of the SBIBD. We require the rows to be presented in lexicographical order for consistency.

We note that only two rows are the same in each of the minimal defining sets for Sets I, II and III: $DI1 = DII1 = DIII1$ and $DI2 = DII2 = DIII2$. Hence the two shareholders holding these rows as shares can participant in any of the three schemes. However as all the other rows of the defining sets are distinct, there being 7+5+5 = 17 of them, any extra row will lead to precisely one of the three designs.

To test the effect of coalitions of $n-1$ (or $n-i$) cheaters trying to collaborate and guess the nth participant's share we have systematically studied the hive of each of the $n-1$ subsets of rows to calculate the ease of guessing.

We found the candidate solutions 11, 18 and 20 for Set I, II and III respectively and we obtained the hives 51, 65 and 80 for Set I, II and III respectively. Hence a coalition of $n-1$ cheaters has to choose $\binom{11}{8}$, $\binom{18}{10}$ and $\binom{20}{10}$ respectively. These give probabilities $1/\binom{11}{8} = 1/165$, $1/\binom{18}{10} = 1/43758$ and $1/\binom{20}{10} = 1/184756$ respectively.

So our results indicate this is not secure enough against cheating.

Another possibility is to give each participant a number of rows. This is explored via the discussion of hives in the Appendix.

Chapter 7

The Size of the Design and Shares

7.1 The Size of the Design

The desirability of lessening the probability of guessing the correct solutions leads us to determine the minimal size of a design for a *secure secret sharing scheme.* The size of the design should not be too small; that would make the probability of guessing the correct solutions high. However the size should not be too large, as possibly this could make the scheme impractical in term of computational implementation.

From the examples of the 2-(16,6,2) design, we can see that this design is still too small to be implemented in a practical secret sharing scheme. The probability of guessing correct solutions is still high ($\geq 1/10^{36}$). The $2-(31,15,7)$ design gave us two choices of defining sets with 13 and 19 vectors as starting rows. However, through experiments of removing rows from the set of starting rows and analysing the probability of guessing the correct solutions, a minimal defining set with 13 rows is better for a secret sharing schemes than a minimal defining set with 19 rows. The probability of guessing the correct solutions for 2-$(31,15,7)$ design is still high; we suggest that in a practical scheme the probability of guessing the correct solutions should be $< 1/10^{40}$.

7.2 The Size of shares

We predicted that, if one row were removed from the starting rows, the number of candidate solutions for a larger size of design will be greater than the number of candidate solutions for a smaller size of design, and this will lead to a smaller probability of guessing the correct solutions. However, examples from previous chapters show that this prediction is not true. From the first example of Example 4.2.1, we see that the number of candidate solutions is only 12 compared to 9 from Example 1.3.2 which is a smaller size design. The probability of guessing the correct solutions in Example 1.3.2

is 0.5×10^{-4} which is greater than the probability of guessing the correct solutions in Example 4.2.1. However, the difference is not significant. This case relates to the need to carefully choose the size of the share for each participant.

The appropriate size of the share is significant in secret sharing schemes. Carefully choosing the size of the share is strongly related to minimising the chance of the opponents being able to cheat successfully. In our experiments, we analysed the influence of removing rows from minimal defining sets on the probability of guessing the correct solutions. Rows removal gives indications of the probability of success of $< n$ shareholder(s) who want to reveal the secret unfairly. As more rows are removed the number of potential solutions grows rapidly. An opponent must guess the correct solutions from the abundant potential solutions. Through experiments, by removing rows one by one we found that removing one row did not give sufficient security against an opponent guessing the correct solutions. This means that it is not desirable to distribute shares of size one row to n *participants*. However, when we removed more than one row we obtained a more appropriate value of the probability. From Table 5.3 and Table 5.5 we see that, by removing more than 2 rows, the probability of being able to cheat successfully became smaller. For the Paley Construction with 13 as minimal defining sets we obtained $1/10^{23}$ as the probability of guessing the correct solutions when we removed 4 rows, this means that if we want use this probability value, we can give 4 rows to 2 participants and 5 rows to one participant as the size of share to each participants. This distribution of shares leads us to $n - out - of - m$ secret sharing schemes as we stated in chapter 3.

The size of share for each participant is also related to the size of the secret itself. Capocelli, de Santis, Gargano and Vacarro [1] and also Jackson, Martin, and O'Keefe [6] state that the size of the shares cannot be less than the size of the secret. This means that we also have to carefully consider the size of the secret in order to have secure secret sharing schemes.

Chapter 8

Conclusion and Discussion

We have shown that the minimal defining set of the $SBIBD(31, 15, 7)$ constructed using the Paley difference set has 13 blocks. We have given a defining set of 19 blocks for the $SBIBD(31, 15, 7)$ constructed using the Hall difference set. We have shown that the smallest minimal defining set for the design constructed from the $Hall$ $(31, 15, 7)$ difference set has more than 15 and less than 19 blocks. We have given a minimal defining set of 19 blocks for the $Hall(31, 15, 7)$.

Finding minimal defining sets of a design and the influence of row removal from its minimal defining set is related to $n-out-of-n$ secret sharing schemes. In $n-out-of-n$ secret sharing schemes, each participant is given a set of rows as a share, and when the participants act together they reveal a confidential message. Opponents, who wish to reveal the message, have less chance of cheating. Further study is needed to consider and investigate the optimal size of a design and shares for practical secure secret sharing schemes in which there is a low probability of guessing the correct solutions. Research is needed to improve the efficiency of the supporting algorithms and the cost of implementation of such secure sharing schemes.

Appendix A

Summary of Numerical Results for 2-(16,6,2) Design

Appendix B

Computer Program Listing

B.1 Generating vectors with size 7

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

int pattrn_0(int);
int pattrn_1(int);
int pattrn_2(int);
int pattrn_3(int);
int pattrn_4(int);
int pattrn_5(int);
int pattrn_6(int);
int pattrn_7(int);
int GetBit(int,unsigned int *);
void SetBit(int,unsigned int *,int);

unsigned int cell[150];

int main()
{

        int i,j,k,maxcol,retValue0,retValue1,retValue2,retValue7,retValue6;
        int retValue3,retValue4,retValue5;
```

```
        i=0;
        maxcol=16;

        retValue0=pattrn_0(i);
        retValue1=pattrn_1(retValue0);
        retValue2=pattrn_2(retValue1);
        retValue3=pattrn_3(retValue2);
        retValue4=pattrn_4(retValue3);
        retValue5=pattrn_5(retValue4);
        retValue6=pattrn_6(retValue5);
        retValue7=pattrn_7(retValue6);


        for(j=0;j<retValue7;j++)
        {
         printf("%d ",j);
         for(k=0;k<maxcol;k++)printf("%d",GetBit(k,&cell[j]));
         printf("\n");
        }

        return 0;

}

int pattrn_0(int rows)
{
        /* pattrn with 0 1's */

        int row,firstRow_0,lastRow_0;

        firstRow_0 = rows;
        lastRow_0 = firstRow_0 + 1;

        for(row=firstRow_0;row<lastRow_0;row++)
        {
```

```
                cell[row]= 0;
        }

        return row;
}

int pattrn_1(int nextRow)
{

        /* pattern with 1 1's */

        int row,i;
        int firstRow_1,lastRow_1;

        firstRow_1 = nextRow;
        lastRow_1 = nextRow + 7;
        i = 0;

        for(row=firstRow_1;row<lastRow_1;row++)
        {
                cell[row]= 0;
                SetBit(i,&cell[row],1);
                i++;
        }

        return row;
}

int pattrn_2(int nextRow)
{

        /* pattern with max 2 1's */

        int row,i,j,k,max2_1;
        int firstRow_2,lastRow_2,maxcol;
```

```
int newPattrn_2,newPattrn_21;

firstRow_2 = nextRow;
lastRow_2 = nextRow + 21;
newPattrn_2 = 1;
newPattrn_21 = 1;
max2_1 = 2;
i = 0;
k = 2;
j = 2;

for(row=firstRow_2;row<lastRow_2;row++)
{

        if(newPattrn_2==1)
        {
                cell[row]= 0;
                for(i=0;i<max2_1;i++)SetBit(i,&cell[row],1);
                i=0;

        }
        if(newPattrn_2==2)
        {
                cell[row]=0;
                SetBit(i,&cell[row],1);

                SetBit(k,&cell[row],1);
                k++;
        }
        if(newPattrn_2==3)
        {
                cell[row]=0;
                if(newPattrn_21==1)
                {
                 for(i=2;i<max2_1+2;i++)SetBit(i,&cell[row],1);
```

```
            }
            if(newPattrn_21==2)
            {
              SetBit(j,&cell[row],1);

              SetBit(k,&cell[row],1);
              k++;

            }
            if(newPattrn_21==3)
            {

              for(i=k;i<max2_1+5;i++)SetBit(i,&cell[row],1);
              SetBit(j,&cell[row],0);
              j++;
            }

}
if(newPattrn_2==3)
{
 if(newPattrn_21==2)
 {
  if(k > 6)
  {
    k=4;j++;
  }
 }
}
if(newPattrn_2==3)
{
 if(newPattrn_21==2)
 {
  if(j > 3)
  {
        j=k=4;newPattrn_21=3;
```

```
                }
               }
              }
               if(newPattrn_2==3)
              {
               if(newPattrn_21==1)
               {
                        i=2;k=4;newPattrn_21=2;
               }
              }

              if(newPattrn_2==2)
              {
               if(k > 6)
               {
                        k=2;i++;
               }
              }
              if(newPattrn_2==2)
              {
               if(i > 1)
               {
                        i=0;k=2;newPattrn_2=3;
               }
              }
              if(newPattrn_2==1)newPattrn_2++;

        }

        return row;
}

int pattrn_3(int nextRow)
{
        int row,i,j,k,m,max3_1;
```

```
int firstRow_3,lastRow_3;
int newPattrn_3,newPattrn_31;

firstRow_3 = nextRow;
lastRow_3 = firstRow_3 + 35;
max3_1 = 3;
newPattrn_3= 1;
newPattrn_31=1;
k = 3;
j = 2;
m = 5;

for(row=firstRow_3;row<lastRow_3;row++)
{
        cell[row]=0;

        if(newPattrn_3 == 1)
        {
         for(i=0;i<max3_1;i++)SetBit(i,&cell[row],1);
        }
        if(newPattrn_3 == 2)
        {
                for(i=0;i<max3_1-1;i++)SetBit(i,&cell[row],1);
                SetBit(j,&cell[row],0);

                SetBit(k,&cell[row],1);
                k++;
                if(k > 6)
                {
                        k=3;j--;
                }
        }
        if(newPattrn_3 == 3)
        {
                for(i=3;i<7;i++)SetBit(i,&cell[row],1);
```

```
                SetBit(j,&cell[row],0);
                j--;
        }
        if(newPattrn_3 == 4)
        {
                SetBit(j,&cell[row],1);

                if(newPattrn_31==1)
                {
                        for(i=3;i<5;i++)SetBit(i,&cell[row],1);
                }
                if(newPattrn_31==2)
                {
                        for(i=5;i<7;i++)SetBit(i,&cell[row],1);
                }
                if(newPattrn_31==3)
                {
                        SetBit(k,&cell[row],1);

                        SetBit(m,&cell[row],1);
                        m++;

                        if(m > 6)
                        {
                            m=5;k++;
                        }

                }

                if(newPattrn_31==1 || newPattrn_31==2)
                {
                        newPattrn_31++;
                }
                if(newPattrn_31==3)
                {
```

```
                        if(k > 4)
                        {
                            k = 3;j++;newPattrn_31=1;
                        }
                    }
                }
                if(newPattrn_3 == 3)
                {
                    if(j < 3 )
                    {
                     j=0;newPattrn_3=4;
                    }

                }

                if(newPattrn_3 == 2)
                {
                    if(j < 0)
                    {
                        j=6;newPattrn_3=3;
                    }
                }
                if(newPattrn_3 == 1)newPattrn_3=2;
        }


        return row;
}

int pattrn_4(int nextRow)
{
        int row,i,j,k,m,max4_1;
        int firstRow_4,lastRow_4,newPattrn_4,newPattrn_41;

        firstRow_4 = nextRow;
```

```
lastRow_4 = firstRow_4 + 35 ;
max4_1 = 4;
newPattrn_4= 1;
newPattrn_41=1;
j = 3;
k = 4;
m = 2;

for(row=firstRow_4;row<lastRow_4;row++)
{
        cell[row]=0;
        if(newPattrn_4==1)
        {
          for(i=0;i<max4_1;i++)SetBit(i,&cell[row],1);
        }
        if(newPattrn_4==2)
        {
          for(i=0;i<max4_1-1;i++)SetBit(i,&cell[row],1);
          SetBit(j,&cell[row],0);

          SetBit(k,&cell[row],1);
          k++;
          if(k > 6)
          {
                j--;k=4;
          }
        }
        if(newPattrn_4==3)
        {
                SetBit(k,&cell[row],1);
                k++;
                for(i=4;i<7;i++)SetBit(i,&cell[row],1);
        }
        if(newPattrn_4==4)
        {
```

```
if(newPattrn_41==1)
{
        for(i=0;i<max4_1-2;i++)SetBit(i,&cell[row],1);
}
if(newPattrn_41==2)
{
        for(i=2;i<max4_1;i++)SetBit(i,&cell[row],1);
}
if(newPattrn_41==3)
{
        SetBit(k,&cell[row],1);

        SetBit(m,&cell[row],1);
}
for(i=4;i<7;i++)SetBit(i,&cell[row],1);
SetBit(j,&cell[row],0);
j--;

if(newPattrn_41==3)
{
        if(j < 4)
        {
         m++;j=6;
        }
        if(m > 3)
        {
          m=2;k++;
        }
}

if(newPattrn_41==1 || newPattrn_41==2)
{
        if(j < 4)
        {
                j=6;newPattrn_41++;
```

```
                        }
                }
        }

        if(newPattrn_4==3)
        {
                if(k > 3)
                {
                        k=0;newPattrn_4=4;
                }
        }
        if(newPattrn_4==2)
        {
                if(j < 0)
                {
                  j = 6;k = 0; newPattrn_4=3;
                }
        }
        if(newPattrn_4==1)newPattrn_4=2;
    }

    return row;
}

int pattrn_5(int nextRow)
{

    int row,i,j,k,max5_1;
    int newPattrn_5,firstRow_5,lastRow_5;

    firstRow_5 = nextRow;
    lastRow_5 = firstRow_5 + 21 ;
    newPattrn_5 = 1;
    max5_1 = 5;
    j = 4;
```

```
k = 5;

        for(row=firstRow_5;row<lastRow_5;row++)
        {
         cell[row]=0;
         if(newPattrn_5==1)
         {
          for(i=0;i<max5_1;i++)SetBit(i,&cell[row],1);
         }
         if(newPattrn_5==2)
         {
          for(i=0;i<max5_1;i++)SetBit(i,&cell[row],1);
          SetBit(j,&cell[row],0);

          SetBit(k,&cell[row],1);
          k++;
          if(k>6)
          {
              j--;k=5;
          }

         }
         if(newPattrn_5==3)
         {
                 for(k=0;k<max5_1-2;k++)SetBit(k,&cell[row],1);

                 for(i=5;i<7;i++)SetBit(i,&cell[row],1);
         }
         if(newPattrn_5==4)
         {
                 SetBit(j,&cell[row],1);
                 for(i=3;i<7;i++)SetBit(i,&cell[row],1);
                 j++;
         }
         if(newPattrn_5==5)
```

```
{
        for(i=0;i<max5_1-3;i++)SetBit(i,&cell[row],1);
        SetBit(j,&cell[row],0);

        SetBit(k,&cell[row],1);
        k++;
        if(k > 4)
        {
          j--;k=3;
        }

        for(i=5;i<7;i++)SetBit(i,&cell[row],1);
}
if(newPattrn_5==5)
{
        if(j < 0)
        {
          j=0;k=3;
        }
}
if(newPattrn_5==4)
{
        if(j > 2)
        {
            j=2;newPattrn_5=5;
        }
}

if(newPattrn_5==3)
{
        newPattrn_5=4;
}
if(newPattrn_5==2)
{
 if(j < 0)
```

```
                {
                        k=3;j=0;newPattrn_5=3;
                }
               }
               if(newPattrn_5==1)
               {
                        newPattrn_5=2;
               }
              }
        return row;
}

int pattrn_6(int nextrows)
{
        /* pattern with max 6 1's */

        int row,i,maxcol;
        int firstRow_6,lastRow_6;

        maxcol = 16;
        firstRow_6 = nextrows;
        lastRow_6 = nextrows + 7;
        i = 0;

        for(row=firstRow_6;row<lastRow_6;row++)
        {
                cell[row]=0;
                cell[row]=~cell[row];

                SetBit(i,&cell[row],0);
                i++;
        }

        return row;
}
```

```
int pattrn_7(int nextRow)
{

        /* pattern with max 7 1's */

        int row;
        int firstRow_7,lastRow_7;

        firstRow_7 = nextRow;
        lastRow_7 = firstRow_7 + 1;


         for(row=firstRow_7;row<lastRow_7;row++)
         {
                 cell[row]=0;
                 cell[row]=~cell[row];
         }

        return row;

}

int GetBit(int index, unsigned int *row)
{
        unsigned int mask = 0x01;
        mask = mask << ((16-index)-1);
        if (*row & mask)
                return 1;
        else return 0;
}

void SetBit(int index, unsigned int *row, int value)
{
        unsigned int mask = 0x01;
```

```
        mask = mask ≪ ((16-index)-1);
        if (value)
        {
                *row = *row | mask;
        }
        else
        {
                mask = ~mask;
                *row = *row & mask;
        }

}
```

B.2 Generating vectors with size 8

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

int pattrn_0(int);
int pattrn_1(int);
int pattrn_2(int);
int pattrn_3(int);
int pattrn_4(int);
int pattrn_5(int);
int pattrn_6(int);
int pattrn_7(int);
int pattrn_8(int);
int GetBit(int,unsigned int *);
void SetBit(int,unsigned int *,int);

unsigned int cell[500];

int main()
```

```
{

        int i,j,k,maxcol,retValue0,retValue1,retValue2,retValue7,retValue6;
        int retValue3,retValue4,retValue5,retValue8;

        i=0;
        maxcol=16;

        retValue0=pattrn_0(i);
        retValue1=pattrn_1(retValue0);
        retValue2=pattrn_2(retValue1);
        retValue3=pattrn_3(retValue2);
        retValue4=pattrn_4(retValue3);
        retValue5=pattrn_5(retValue4);
        retValue6=pattrn_6(retValue5);
        retValue7=pattrn_7(retValue6);
        retValue8=pattrn_8(retValue7);

        for(j=retValue3;j<retValue4;j++)
        {
         printf("%d ",j);
         for(k=0;k<maxcol;k++)printf("%d",GetBit(k,&cell[j]));
         printf("\n");
        }

        return 0;

}

int pattrn_0(int rows)
{
        /* pattrn with 0 1's */

        int row,firstRow_0,lastRow_0;
```

```
        firstRow_0 = rows;
        lastRow_0 = firstRow_0 + 1;

        for(row=firstRow_0;row<lastRow_0;row++)
        {
                cell[row]= 0;
        }

        return row;
}


int pattrn_1(int nextRow)
{

        /* pattern with 1 1's */

        int row,i;
        int firstRow_1,lastRow_1;

        firstRow_1 = nextRow;
        lastRow_1 = nextRow + 8;
        i = 0;

        for(row=firstRow_1;row<lastRow_1;row++)
        {
                cell[row]= 0;
                SetBit(i,&cell[row],1);
                i++;
        }

        return row;
}

int pattrn_2(int nextRow)
```

```
{
        int row,i,j,k,max2_1;
        int firstRow_2,lastRow_2,newPattrn_2,newPattrn_21;

        firstRow_2 = nextRow;
        lastRow_2 = firstRow_2 + 28;
        max2_1 = 2;
        newPattrn_2 = 1;
        newPattrn_21= 1;
        j = 2;
        k = 0;

        for(row=firstRow_2;row<lastRow_2;row++)
        {
                cell[row]=0;
                if(newPattrn_2==1)
                {
                        for(i=0;i<max2_1;i++)SetBit(i,&cell[row],1);
                }
                if(newPattrn_2==2)
                {
                        SetBit(k,&cell[row],1);

                        SetBit(j,&cell[row],1);
                        j++;
                        if(j > 7)
                        {
                            j = 2; k++;
                        }
                }
                if(newPattrn_2==3)
                {
                        if(newPattrn_21==1)
                        {
                                for(i=2;i<4;i++)SetBit(i,&cell[row],1);
```

```
            }
            if(newPattrn_21==2)
            {
                    SetBit(j,&cell[row],1);

                    SetBit(k,&cell[row],1);
                    k++;
                    if(k > 7)
                    {
                        k=4;j++;
                    }
            }
            if(newPattrn_21==3)
            {
                    for(i=k;i<k+2;i++)SetBit(i,&cell[row],1);
                    k++;
            }
            if(newPattrn_21==4)
            {
                    SetBit(k,&cell[row],1);

                    SetBit(j,&cell[row],1);
                    j++;
                    if(j > 7)
                    {
                        k++;j = 7;
                    }
            }
    }

     if(newPattrn_2==3)
     {
             if(newPattrn_21==3)
             {
                     if(k > 6)
```

```
                                {
                                    k = 4;newPattrn_21=4;
                                }
                            }
                            if(newPattrn_21==2)
                            {
                                if(j > 3)
                                {
                                  j = 6; k = 4;newPattrn_21=3;
                                }
                            }

                            if(newPattrn_21==1)newPattrn_21=2;
                        }

                    if(newPattrn_2==2)
                    {
                            if(k > 1)
                            {
                                k=4;j=2;newPattrn_2=3;
                            }
                    }
                    if(newPattrn_2==1)newPattrn_2=2;
            }

            return row;
}

int pattrn_3(int nextRow)
{
            int row,i,j,k,m,n;
            int firstRow_3,lastRow_3,newPattrn_3;
            int max3_1,newPattrn_31,newPattrn_32;

            firstRow_3 = nextRow;
```

```
lastRow_3 = firstRow_3 + 56;
newPattrn_3 = 1;
newPattrn_31 = 1;
newPattrn_32 = 1;
max3_1 = 3;
j = 3;
k = 2;
m = 5;
n = 7;

for(row=firstRow_3;row<lastRow_3;row++)
{
        cell[row]=0;
        if(newPattrn_3==1)
        {
                for(i=0;i<max3_1;i++)SetBit(i,&cell[row],1);
        }
        if(newPattrn_3==2)
        {
                for(i=0;i<max3_1;i++)SetBit(i,&cell[row],1);
                SetBit(k,&cell[row],0);

                SetBit(j,&cell[row],1);
                j++;
                if(j > 7)
                {
                    k--;j = 3;
                }
        }
        if(newPattrn_3==3)
        {
                SetBit(k,&cell[row],1);

                if(newPattrn_31==1)
                {
```

```
        for(i=3;i<max3_1+2;i++)SetBit(i,&cell[row],1);
    }
    if(newPattrn_31==2)
    {
            SetBit(j,&cell[row],1);

            SetBit(m,&cell[row],1);
            m++;
            if(m > 7)
            {
              j++; m = 5;
            }
    }
    if(newPattrn_31==3)
    {
            for(i=5;i<max3_1+5;i++)SetBit(i,&cell[row],1);

            SetBit(n,&cell[row],0);
            n--;
    }

    if(newPattrn_31==2)
    {
            if(j > 4)
            {
                j = 3;m = 5;newPattrn_31=3;
            }
    }
    if(newPattrn_31==1)newPattrn_31=2;
    if(n < 5)
    {
          n = 7;newPattrn_31=1;k++;
    }

}
```

```
if(newPattrn_3==4)
{
        if(newPattrn_32==1)
        {
         for(i=3;i<max3_1+3;i++)SetBit(i,&cell[row],1);
        }
        if(newPattrn_32==2)
        {
         for(i=3;i<max3_1+2;i++)SetBit(i,&cell[row],1);
         SetBit(j,&cell[row],0);

         SetBit(k,&cell[row],1);
         k++;
         if(k > 7)
         {
           j--; k = 6;
         }

        }
        if(newPattrn_32==3)
        {
                SetBit(k,&cell[row],1);
                k++;

                for(i=6;i<8;i++)SetBit(i,&cell[row],1);

        }
        if(newPattrn_32==2)
        {
                if(j < 3)
                {
                 newPattrn_32=3;k = 3;
                }
        }
        if(newPattrn_32==1)newPattrn_32=2;
```

```
                }
                if(newPattrn_3==3)
                {
                        if(k > 2)
                        {
                            k = 6;j = 5;newPattrn_3=4;
                        }
                }
                if(newPattrn_3==2)
                {
                        if(k < 0)
                        {
                          k = 0;j = 3;newPattrn_3=3;
                        }
                }
                if(newPattrn_3==1) newPattrn_3=2;
        }

        return row;
}


int pattrn_4(int nextRow)
{
        int row,i,j,k,k1,j1,max4_1,last_pattrn;
        int firstRow_4,lastRow_4,newPattrn_4,newPattrn_41,newPattrn_42;


        firstRow_4 = nextRow;
        lastRow_4 = firstRow_4 + 70;
        newPattrn_4= 1;
        newPattrn_41= 1;
        newPattrn_42= 1;
        last_pattrn = 0;
```

```
max4_1 = 4;
j = 3;
k = 4;
j1 = 6;
k1 = 4;

for(row=firstRow_4;row<lastRow_4;row++)
{

        cell[row]=0;
        if(newPattrn_4==1)
        {
                for(i=0;i<max4_1;i++)SetBit(i,&cell[row],1);
        }
        if(newPattrn_4==2)
        {
                for(i=0;i<max4_1;i++)SetBit(i,&cell[row],1);
                SetBit(j,&cell[row],0);

                SetBit(k,&cell[row],1);
                k++;
                if(k > 7)
                {
                    k = 4;j--;
                }
        }
        if(newPattrn_4==3)
        {
                SetBit(j,&cell[row],1);

                for(i=4;i<max4_1+4;i++)SetBit(i,&cell[row],1);
                SetBit(k,&cell[row],0);
                k--;
                if(k < 4)
                {
```

```
                j++; k = 7;
            }
        }
        if(newPattrn_4==4)
        {
            for(i=4;i<max4_1+4;i++)SetBit(i,&cell[row],1);
        }
        if(newPattrn_4==5)
        {
            if(newPattrn_41==1)
            {
             for(i=0;i<2;i++)SetBit(i,&cell[row],1);
            }
            if(newPattrn_41==2)
            {
             SetBit(k,&cell[row],1);

             SetBit(j,&cell[row],1);
            }

            if(newPattrn_41==3)
            {
              for(i=0;i<2;i++)SetBit(i,&cell[row],0);
              for(i=2;i<4;i++)SetBit(i,&cell[row],1);
            }



            if(newPattrn_42==1)
            {
             for(i=4;i<6;i++)SetBit(i,&cell[row],1);
            }
            if(newPattrn_42==2)
            {
             SetBit(k1,&cell[row],1);
```

```
 SetBit(j1,&cell[row],1);
 j1++;
 if(j1 > 7)
 {
   k1++; j1 =6;
 }
}
if(newPattrn_42==3)
{
        for(i=6;i<8;i++)SetBit(i,&cell[row],1);
        last_pattrn = 1;
}


if(newPattrn_42==2)
{
   if(k1 > 5)
   {
        k1=4;newPattrn_42=3;
   }
}
if(newPattrn_42==1)newPattrn_42=2;

if(newPattrn_41==2)
{
        if(newPattrn_42==3 && last_pattrn==1)
        {
         j++;newPattrn_42=1;
         last_pattrn = 0;
         if(j > 3)
         {
            j = 2;k++;
         }
        }
```

```
                    if(k > 1)
                    {

                      if(newPattrn_42==3 && last_pattrn==1)
                      {
                       newPattrn_42=1;
                       last_pattrn = 0; k=0;j=0;
                      }
                      newPattrn_41=3;

                    }

            }


            if(newPattrn_41==1)
            {
             if(newPattrn_42==3 && last_pattrn==1)
             {
                    newPattrn_41=2;newPattrn_42=1;
                    last_pattrn = 0;k=0;j=2;
             }
            }

}
if(newPattrn_4==4)
{
 j = 0;k =2;newPattrn_4=5;
}
if(newPattrn_4==3)
{
            if(j > 3)
            {
               j = 0; newPattrn_4=4;
            }
```

```
                }
                if(newPattrn_4==2)
                {
                        if(j < 0)
                        {
                          newPattrn_4=3; j = 0;k = 7;
                        }
                }
                if(newPattrn_4==1)newPattrn_4=2;
        }
        return row;
}



int pattrn_5(int nextRow)
{
        int row,i,j,k,m,max5_1;
        int firstRow_5,lastRow_5;
        int newPattrn_5,newPattrn_51,newPattrn_52;

        firstRow_5 = nextRow;
        lastRow_5 = firstRow_5 + 56;
        max5_1 = 5;
        newPattrn_5= 1;
        newPattrn_51= 1;
        newPattrn_52= 1;
        j = 4;
        k = 5;
        m = 7;

        for(row=firstRow_5;row<lastRow_5;row++)
        {
                cell[row]=0;
                if(newPattrn_5==1)
```

```
{
    for(i=0;i<max5_1;i++)SetBit(i,&cell[row],1);
}
if(newPattrn_5==2)
{
        for(i=0;i<max5_1;i++)SetBit(i,&cell[row],1);
        SetBit(j,&cell[row],0);

        SetBit(k,&cell[row],1);
        k++;
        if(k > 7)
        {
           j--;k=5;
        }
}
if(newPattrn_5==3)
{
        if(newPattrn_51==1)
        {
                for(i=0;i<max5_1-2;i++)SetBit(i,&cell[row],1);
        }
        if(newPattrn_51==2)
        {
                for(i=0;i<max5_1-3;i++)SetBit(i,&cell[row],1);
                SetBit(j,&cell[row],0);

                SetBit(k,&cell[row],1);

        }
        if(newPattrn_51==3)
        {
                SetBit(j,&cell[row],1);
                for(i=3;i<max5_1;i++)SetBit(i,&cell[row],1);
        }
        for(i=5;i<8;i++)SetBit(i,&cell[row],1);
```

```
                SetBit(m,&cell[row],0);
                m--;
                if(m < 5)
                {
                    if(newPattrn_51==3)
                    {
                        j++; m=7;

                    }
                    if(newPattrn_51==2)
                    {
                        m = 7;k++;
                        if(k > 4)
                        {
                            k=3;j--;
                        }
                        if(j < 0)
                        {
                            j = 0;newPattrn_51=3;
                        }
                    }

                    if(newPattrn_51==1)
                    {
                        newPattrn_51=2;m = 7;
                    }

                }

            }
            if(newPattrn_5==4)
            {
                if(newPattrn_52==1)
                {
                    for(i=0;i<max5_1-3;i++)SetBit(i,&cell[row],1);
```

```
        }
        if(newPattrn_52==2)
        {
                SetBit(k,&cell[row],1);

                SetBit(j,&cell[row],1);
                j++;
                if(j > 4)
                {
                    k++;j=2;
                }
        }
        if(newPattrn_52==3)
        {
                for(i=2;i<max5_1-1;i++)SetBit(i,&cell[row],1);
                SetBit(j,&cell[row],0);
                j--;
        }
        for(i=5;i<8;i++)SetBit(i,&cell[row],1);

        if(newPattrn_52==2)
        {
                if(k > 1)
                {
                    k = 0;j = 4;newPattrn_52=3;
                }
        }
        if(newPattrn_52==1)newPattrn_52=2;

}
if(newPattrn_5==3)
{
 if(newPattrn_51==3)
 {
        if(j > 2)
```

```
                    {
                        j=2;k=0;newPattrn_5=4;
                    }
                }
            }
            if(newPattrn_5==2)
            {
                if(j<0)
                {
                  k=3;j=2;newPattrn_5=3;
                }
            }
            if(newPattrn_5==1)newPattrn_5=2;
        }
        return row;
}

int pattrn_6(int nextRow)
{
        int row,i,j,k,max6_1;
        int firstRow_6,lastRow_6;
        int newPattrn_6,newPattrn_61;

        firstRow_6 = nextRow;
        lastRow_6 = firstRow_6 + 28;
        max6_1 = 6;
        newPattrn_6 = 1;
        newPattrn_61= 1;
        j = 5;
        k = 6;

        for(row=firstRow_6;row≤lastRow_6;row++)
        {
            cell[row]=0;
            if(newPattrn_6==1)
```

```
{
        for(i=0;i<max6_1;i++)SetBit(i,&cell[row],1);
}
if(newPattrn_6==2)
{
        for(i=0;i<max6_1;i++)SetBit(i,&cell[row],1);
        SetBit(j,&cell[row],0);

        SetBit(k,&cell[row],1);
        k++;
        if(k > 7)
        {
          j--; k=6;
        }
}
if(newPattrn_6==3)
{
        if(newPattrn_61==1)
        {
                for(i=0;i<max6_1-2;i++)SetBit(i,&cell[row],1);
        }
        if(newPattrn_61==2)
        {
                for(i=0;i<max6_1-2;i++)SetBit(i,&cell[row],1);
                SetBit(j,&cell[row],0);

                SetBit(k,&cell[row],1);
                k++;
                if(k > 5)
                {
                    k=4;j--;
                }
        }
        if(newPattrn_61==3)
        {
```

```
                for(i=j;i<j+2;i++)SetBit(i,&cell[row],1);
                j++;

                for(i=4;i<6;i++)SetBit(i,&cell[row],1);
        }
        if(newPattrn_61==4)
        {
                SetBit(j,&cell[row],1);

                SetBit(k,&cell[row],1);
                k++;
                if(k > 3)
                {
                   j++;k=3;
                }
                for(i=4;i<6;i++)SetBit(i,&cell[row],1);

        }
        if(newPattrn_61==3)
        {
                if(j > 3)
                {
                   j=0;k=2;newPattrn_61=4;
                }
        }
        if(newPattrn_61==2)
        {
                if( j < 0)
                {
                        j = 0;newPattrn_61=3;
                }
        }
        if(newPattrn_61==1)newPattrn_61=2;
        for(i=6;i<8;i++)SetBit(i,&cell[row],1);
}
```

```
                if(newPattrn_6==2)
                {
                        if(j < 0)
                        {
                          j = 3;k = 4;newPattrn_6=3;
                        }
                }
                if(newPattrn_6==1)newPattrn_6=2;
        }
        return row;
}

int pattrn_7(int nextRow)
{
        int row,i;
        int firstRow_7,lastRow_7;

        firstRow_7 = nextRow;
        lastRow_7 = firstRow_7 + 8;
        i = 0;

        for(row=firstRow_7;row<lastRow_7;row++)
        {
                cell[row]=0;
                cell[row]=~cell[row];

                SetBit(i,&cell[row],0);
                i++;
        }
        return row;

}


int pattrn_8(int nextRow)
```

```
{

        /* pattern with max 8 1's */

        int row;
        int firstRow_8,lastRow_8;

        firstRow_8 = nextRow;
        lastRow_8 = firstRow_8 + 1;;


         for(row=firstRow_8;row<lastRow_8;row++)
         {
                cell[row]=0;
                cell[row]=~cell[row];
         }

        return row;

}

int GetBit(int index, unsigned int *row)
{
        unsigned int mask = 0x01;
        mask = mask << ((16-index)-1);
        if (*row & mask)
                return 1;
        else return 0;
}

void SetBit(int index, unsigned int *row, int value)
{
        unsigned int mask = 0x01;
        mask = mask << ((16-index)-1);
        if (value)
```

```
        {
                *row = *row | mask;
        }
        else
        {
                mask = ~mask;
                *row = *row & mask;
        }

}
```

B.3 Finding Candidate Solutions

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

void inner_product(int);


FILE *finput1, *finput2, *finput3;

int i,j,k,col,newRow;
int max_i,max_k, maxcol, max_inp, scount;
int cellinp[30][32], cellint[100][32], outputvector[100][32];
char cell15[10000][32],cell16[15000][32],cell[100][32], inputvector[30][32];
char *resinp,input1[81],input2[81] ,input3[81];

main(int argc, char **argv)
{

        maxcol = 31;
        scount = 0;
```

```
/* Open and read file 1 */

if((finput1=fopen(argv[1],"r"))==NULL)
{
        fprintf(stderr,"%s:cannot open%s \n",argv[0],argv[1]);
        exit(1);
}

i = 1;
while((resinp=fgets(input1,80,finput1))≠NULL)
{
        for(j=0;j<15;j++)cell15[i][j]=input1[j];
        i++;
}

max_i = i-1;

printf(" i %d \n",max_i);
fclose(finput1);

/* Open and read file 2 */

if((finput2=fopen(argv[2],"r"))==NULL)
{
        fprintf(stderr,"%s:cannot open \n",argv[0],argv[2]);
        exit(1);
}

k = 1;
while((resinp=fgets(input2,80,finput2))≠NULL)
{
        for(j=0;j<16;j++)cell16[k][j]=input2[j];
        k++;
}
```

```
max_k = k-1;

printf(" k %d \n",max_k);
fclose(finput2);

/* Open and read file 3 */

if((finput3=fopen(argv[3],"r"))==NULL)
{
        fprintf(stderr,"%s:cannot open \n",argv[0],argv[3]);
        exit(1);
}

k = 1;
while((resinp=fgets(input3,80,finput3))≠NULL)
{
        strncpy(inputvector[k],input3,31);
        for(j=0;j<maxcol;j++)
        {
         if(input3[j]=='0')cellinp[k][j]=0;
         else cellinp[k][j]=1;
        }
        k++;
}

max_inp = k-1;

printf(" inp %d \n",max_inp);
fclose(finput3);

i = 1;
newRow = 1;
k = 1;

while(i ≤ max_i)
```

```
{
        for(j=0;j<15;j++) cell[newRow][j]=cell15[i][j];

        for(j=15;j<31;j++)cell[newRow][j]=cell16[k][j-15];

        for(col=0;col<maxcol;col++)
        {
         if(cell[newRow][col]=='0')cellint[newRow][col]=0;
         else cellint[newRow][col]=1;
        }

        inner_product(newRow);

        k++;

                if(k > max_k)
                {
                        i++;k=1;
                }


}

if(scount > 0)
{
        for(i=1;i≤scount;i++)
        {
            printf("%d.",i);
            for(j=0;j<maxcol;j++)printf("%d",outputvector[i][j]);
            printf("\n");
        }
}

scount = 0;
return 0;
```

```
}


void inner_product( int k )
{
  int kk,jj,ii,count,truth,iprod,cond;

  truth=0;

  for(kk=1;kk≤max_inp;kk++)
  {
   if((cond=strncmp(cell[k],inputvector[kk],maxcol))==0)
   {
    truth=1;
    break;
   }
  }

  if(truth==0)
  {
   count=0;
   for(kk=1;kk≤max_inp;kk++)
   {
    iprod=0;
    for(jj=0;jj<maxcol;jj++)
    {
     iprod=iprod + ( cellint[k][jj]*cellinp[kk][jj]);
    }
    if(iprod==7)
    {
     count++;
    }
    else break;
```

```
    }
    if(count==max_inp)
     {

      scount++;

      for(ii=0;ii<maxcol;ii++) outputvector[scount][ii]=cellint[k][ii];

     }

   }
   return;
}
```

B.4 Finding Candidate Solutions with Permutation of Starting Rows

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

void inner_product(int);


FILE *finput1, *finput2, *finput3;

int i,j,k,col,aa,cc,xx,zz,newRow;
int k1,k2,k3,k4,k5,k6,k7,k8,k9,k10,k11,k12,k13,k14;
int max_i,max_k, maxcol, max_inp, scount,comb_inp[15][32];
int cellinp[20][32], cellint[100][32], outputvector[100][32];
char cell15[10000][32],cell16[15000][32],cell[100][32], inputvector[20][32];
char *resinp,input1[81],input2[81] ,input3[81];
char comb_inps[15][32];
```

```
main(int argc, char **argv)
{

        if(argc ≠ 4)
        {
                printf("Please add your files \n");
                exit(1);
        }

        maxcol = 31;
        scount = 0;

/* Open and read file 1 */

 if((finput1=fopen(argv[1],"r"))==NULL)
 {
        fprintf(stderr,"%s:cannot open%s \n",argv[0],argv[1]);
        exit(1);
 }

 i = 1;
 while((resinp=fgets(input1,80,finput1))≠NULL)
 {
        for(j=0;j<15;j++)cell15[i][j]=input1[j];
        i++;
 }

 max_i = i-1;

 printf(" i %d \n",max_i);
 fclose(finput1);

/* Open and read file 2 */

 if((finput2=fopen(argv[2],"r"))==NULL)
```

```
{
        fprintf(stderr,"%s:cannot open \n",argv[0],argv[2]);
        exit(1);
}

k = 1;
while((resinp=fgets(input2,80,finput2))≠NULL)
{
        for(j=0;j<16;j++)cell16[k][j]=input2[j];
        k++;
}

max_k = k-1;

printf(" k %d \n",max_k);
fclose(finput2);

/* Open and read file 3 */

if((finput3=fopen(argv[3],"r"))==NULL)
{
        fprintf(stderr,"%s:cannot open \n",argv[0],argv[3]);
        exit(1);
}

k = 1;
while((resinp=fgets(input3,80,finput3))≠NULL)
{
        strncpy(inputvector[k],input3,31);
        for(j=0;j<maxcol;j++)
        {
         if(input3[j]=='0')cellinp[k][j]=0;
         else cellinp[k][j]=1;
        }
        k++;
```

```
}

max_inp = k-1;

printf(" inp %d \n",max_inp);
fclose(finput3);

for(k1=1;k1≤3;k1++){

 for(k2=k1+1;k2≤4;k2++){

  for(k3=k2+1;k3≤5;k3++){

   for(k4=k3+1;k4≤6;k4++){

    for(k5=k4+1;k5≤7;k5++){

     for(k6=k5+1;k6≤8;k6++){

      for(k7=k6+1;k7≤9;k7++){

       for(k8=k7+1;k8≤10;k8++){

        for(k9=k8+1;k9≤11;k9++){

         for(k10=k9+1;k10≤12;k10++){

          for(k11=k10+1;k11≤13;k11++){

           for(k12=k11+1;k12≤14;k12++){

            for(k13=k12+1;k13≤15;k13++){

             strncpy(comb_inps[1],inputvector[k1],31);
             strncpy(comb_inps[2],inputvector[k2],31);
```

```
strncpy(comb_inps[3],inputvector[k3],31);
strncpy(comb_inps[4],inputvector[k4],31);
strncpy(comb_inps[5],inputvector[k5],31);
strncpy(comb_inps[6],inputvector[k6],31);
strncpy(comb_inps[7],inputvector[k7],31);
strncpy(comb_inps[8],inputvector[k8],31);
strncpy(comb_inps[9],inputvector[k9],31);
strncpy(comb_inps[10],inputvector[k10],31);
strncpy(comb_inps[11],inputvector[k11],31);
strncpy(comb_inps[12],inputvector[k12],31);
strncpy(comb_inps[13],inputvector[k13],31);

 for(zz=0;zz<31;zz++)
{
 comb_inp[1][zz]=cellinp[k1][zz];

 comb_inp[2][zz]=cellinp[k2][zz];

 comb_inp[3][zz]=cellinp[k3][zz];

 comb_inp[4][zz]=cellinp[k4][zz];

 comb_inp[5][zz]=cellinp[k5][zz];

 comb_inp[6][zz]=cellinp[k6][zz];

 comb_inp[7][zz]=cellinp[k7][zz];

 comb_inp[8][zz]=cellinp[k8][zz];

 comb_inp[9][zz]=cellinp[k9][zz];

 comb_inp[10][zz]=cellinp[k10][zz];

 comb_inp[11][zz]=cellinp[k11][zz];
```

```
                comb_inp[12][zz]=cellinp[k12][zz];

                comb_inp[13][zz]=cellinp[k13][zz];

               }

i = 1;
newRow = 1;
k = 1;

while(i ≤ max_i)
{
        for(j=0;j<15;j++) cell[newRow][j]=cell15[i][j];

        for(j=15;j<31;j++)cell[newRow][j]=cell16[k][j-15];

        for(col=0;col<maxcol;col++)
        {
         if(cell[newRow][col]=='0')cellint[newRow][col]=0;
         else cellint[newRow][col]=1;
        }

        inner_product(newRow);

        k++;
                if(k > max_k)
                {
                        i++;k=1;
                }


 }

        if(scount > 0)
```

```
        {
                printf(" Combination of inputs :  %d %d %d %d %d %d %d
%d %d %d %d %d %d \n", k1,k2,k3,k4,k5,k6,k7,k8,k9,k10,k11,k12,k13);


        /* for(xx=1;xx<=max_inp-1;xx++)
        {
                printf(" [%d] :",xx);
                for(zz=0;zz<maxcol;zz++)
                {
                        printf(" %d ",comb_inp[xx][zz]);
                }
                printf("\n");
        }*/

        printf("\n");
        printf("Candidate Solutions :  \n");
        printf("\n");
         for(cc=1;cc≤scount;cc++)
         {
                printf(" %d.   :",cc);
                for(aa=0;aa<maxcol;aa++)
                {
                        printf("%d",outputvector[cc][aa]);
                }
                printf(" \n");
         }
        }
        scount = 0;
}
}
}
}
}
}
```

```
}
}
}
}
}
}
}

 return 0;

}

void inner_product( int k )
{
  int kk,jj,ii,count,truth,iprod,cond;

  truth=0;

  for(kk=1;kk≤max_inp-2;kk++)
  {
   if((cond=strncmp(cell[k],comb_inps[kk],maxcol))==0)
   {
    truth=1;
    break;
   }
  }

  if(truth==0)
  {
   count=0;
   for(kk=1;kk≤max_inp-2;kk++)
   {
    iprod=0;
    for(jj=0;jj<maxcol;jj++)
```

```
    {
     iprod=iprod + ( cellint[k][jj]*comb_inp[kk][jj]);
    }
    if(iprod==7)
    {
     count++;
    }
    else break;
   }
   if(count==max_inp-2)
    {

     scount++;

     for(ii=0;ii<maxcol;ii++) outputvector[scount][ii]=cellint[k][ii];

    }

  }
  return;
}
```

Bibliography

[1] R.M. Capocelli, A.de Santis, L. Gargano, and U. Vacarro, *On the size of shares for secret sharing schemes.* Journal of Cryptology, 1993, pp.157-167.

[2] L F Fitina, Jennifer Seberry and Ghulam R Chaudhry, Back circulant Latin squares and the influence of a set, *Australasian J Combin.,* (submitted)

[3] C.S. Greenhill and A.P.Street, *Smallest defining sets of some small t-designs and relations to the Petersen graph.* Australiasian Journal of Combinatorics, 1998, pp. 7-17.

[4] Marshall Hall Jr, *Combinatorial Theory,* Blaisdell, Waltham, Mass. 1967.

[5] Q.M. Husain. *On the totally of the solutions for the symmetrial incomplete block design :* $\lambda = 2, k = 5$ *or* 6. Sankhya, 7 (1945), pp. 204-207.

[6] W.A. Jackson, K.M. Martin, and C.M. O'Keefe, *Ideal secret sharing schemes with multiple secrets.* Journal of Cryptology, 1996, pp. 233-250.

[7] J.A. John, *Cyclic Designs.* Chapman and Hall, London, 1987.

[8] T. Kunkle and J. Seberry, *A few more small defining sets for SBIBD(4t-1,2t-1,t-1).* Bulletin of the ICA, vol 12, 1994, pp. 61-64.

[9] H.J. Ryser. *Combinatorial Mathematics.* John Wiley and Sons, New Jersey, 1963.

[10] D. Sarvate and J. Seberry, *A note on small defining sets for some SBIBD(4t-1,2t-1,t-1).* Bulletin of the ICA, vol 10, 1994, pp. 26-32.

[11] A. Shamir. *How to share a secret.* Communications of the ACM, vol 22, Nov.1979, pp. 612-613.

[12] J. Seberry. *On small defining sets for some SBIBD(4t-1,2t-1,t-1).* Bulletin of the ICA, vol 4, 1992, pp. 58-62.

[13] G.J. Simmons. *Robust shared secret schemes or 'How to be sure you have the Right Answer Even Though You don't Know the Question.* Congressus Numerantum,vol 68, 1989, pp215-248.

[14] D.R. Stinson. *Cryptography : Theory and Practice.* CRC Press, Boca Raton, 1995.

[15] A.P.Street and D.J.Street *Combinatorics of Experimental Design.* Oxford Science Publications, Oxford, 1987.

[16] J.Utami and J.Seberry *on small defining sets for SBIBD(31,15,7)*, to be submitted for publication.